Stat 274 Pset 4

Karl Jiang

November 20, 2016

1) Speaking in monotone

a)

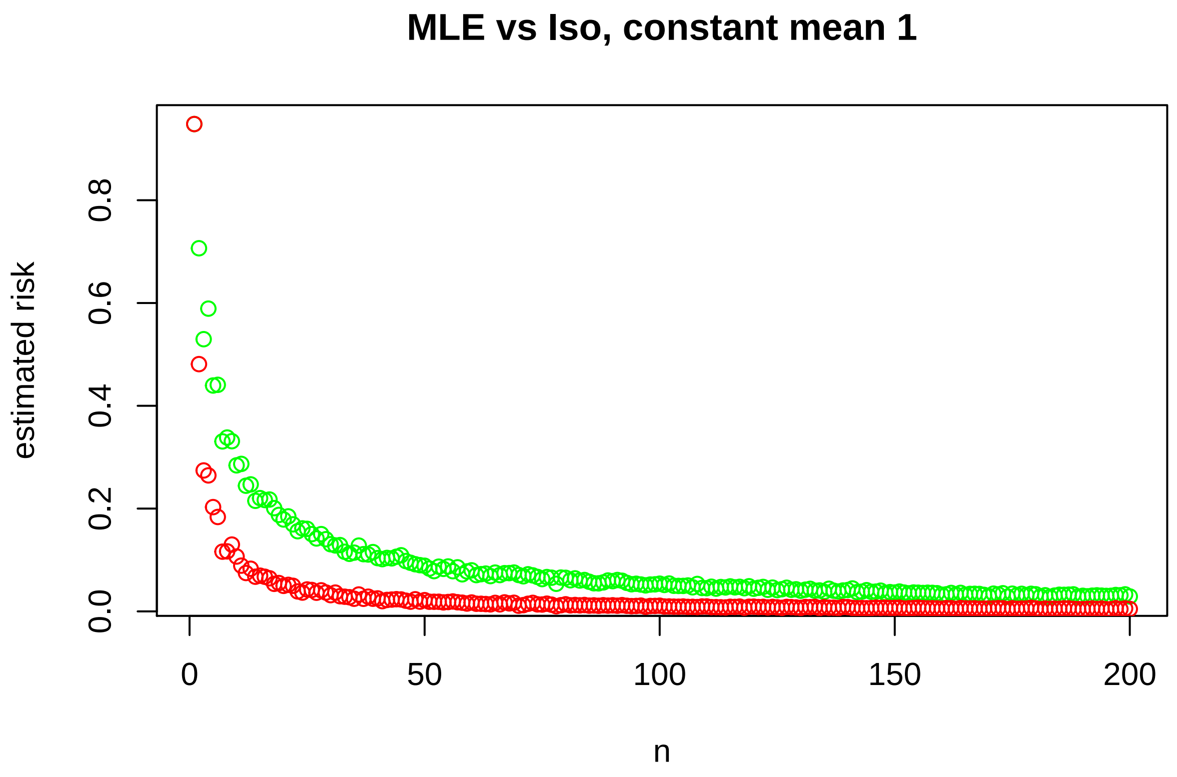
Let our samples be $Z_i \sim N(\theta_i, 1^2)$, where choose all $\theta_i = 1$

MLE Risk)

```
#install.packages("Iso")
library(Iso)
```

```
## Iso 0.0-17
```

```
plot_estimated_risk <- function(N, sigma_n, sims, thetas_f, plot = TRUE, title = ""){
  approx_risks_sims <- c()
  mle_risks_sims <- c()

  for(n in N){
    approx_risk_sim <- c()
    mle_risk_sim <- c()
    for(i in 1:sims){
      thetas = thetas_f(n)
      data <- rnorm(n, mean = thetas, sd = sigma_n)
      pavae <- pava(data)
      mle <- mean(data)

      approx_risk <- mean((pavae - thetas)^2)
      approx_risk_sim <- c(approx_risk_sim, approx_risk)

      mle_risk_sim <- c(mle_risk_sim, (mle - thetas)^2)
    }
    approx_risks_sims <- c(approx_risks_sims, mean(approx_risk_sim) )
    mle_risks_sims <- c(mle_risks_sims, mean(mle_risk_sim))
  }

  if(plot){
  #red is mle, green is pava
  plot(approx_risks_sims, col = "green", xlab = "n", ylab = "estimated risk", main = title)
  lines(mle_risks_sims, col = "red", type = "p")
  }
  return(approx_risks_sims)
}

thetas_1 <- function(n){
  return(1)
}
N <- seq(1, 200)
sims <- 200
sigma_n <- 1
a <- plot_estimated_risk(N = N, sigma_n = sigma_n, sims = sims, thetas_f = thetas_1,
                         title = "MLE vs Iso, constant mean 1")
```



The MLE estimator is red, the isotonic regression estimator is green. The MLE performs better than isotonic regression, which makes sense. Since our data is given by $X_1, ..., X_n \sim N(\theta, \sigma^2)$, the risk of the MLE is unbiased: $\mathbb{E}[\bar{X}] = \theta$ and the variance is $var(\bar{X}) = var(\frac{\sum_{i=1}^{n} X_i}{n}) = \frac{nvar(X_i)}{n^2} = \frac{\sigma^2}{n} = \frac{1}{n}$. This converges much faster than isotonic regression which is given by $O(\frac{logn}{n})$

b)

setting 1: 1st half 0, 2nd half 1

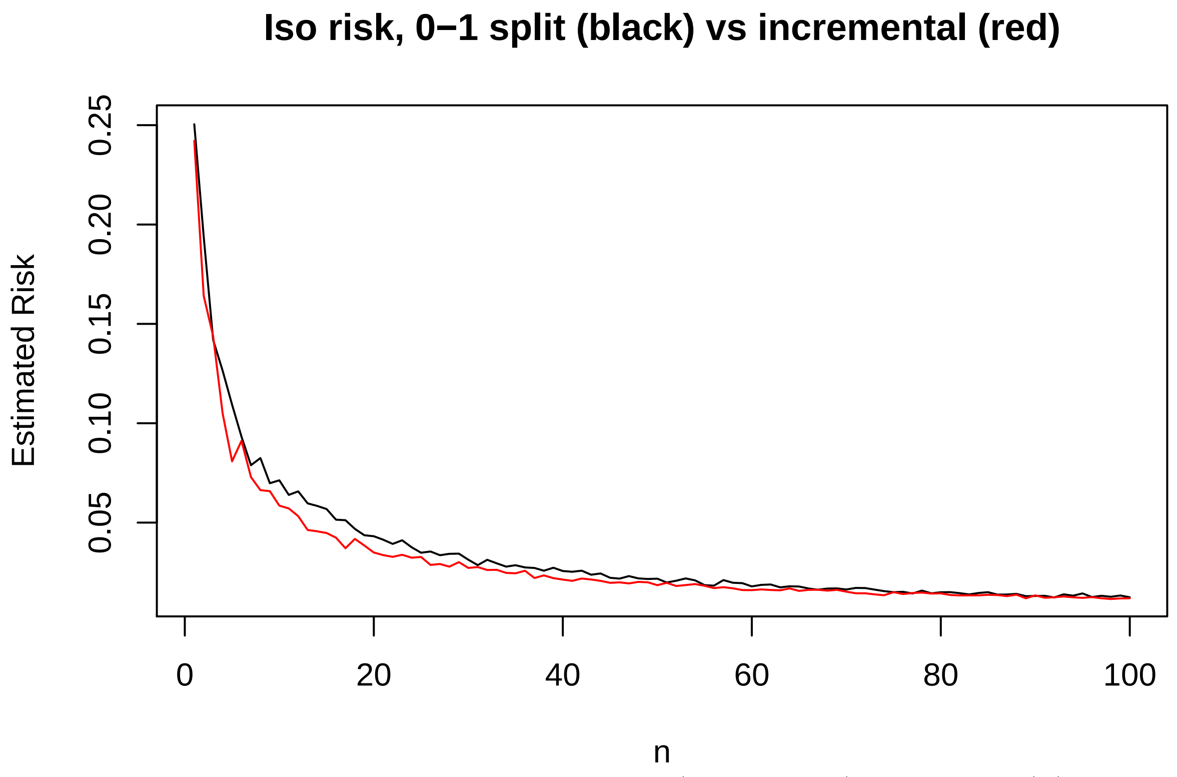
```
N <- seq(2, 200, 2)
sigma_n <- 1/2
#sims <- 200
thetas_half <- function(n) {
  return(c(rep(0, n/2), rep(1, n/2 ) ) )
}

b <- plot_estimated_risk(N, sigma_n, sims, thetas_f = thetas_half, FALSE)
```

setting 2: 0 to 1 increments

```
#sims <- 200
thetas_incr <- function(n){
  return(seq(0, 1, length.out = n))
}
c <- plot_estimated_risk(N, sigma_n, sims, thetas_incr, FALSE)
```

Comparison between both settings

```
#Comparison
plot(b, main = "Iso risk, 0-1 split (black) vs incremental (red)", xlab = "n", ylab = "Estimated Risk",
lines(c, col = "red")
```



Isotonic Regression gives similar Risks across seetings 1 (split means (0 and 1, black)) and setting 2: incrementally increasing means as n gets large (red), though the split means seems have . Probably because the assumption of monotonicity is true in these two cases.

3) Motor

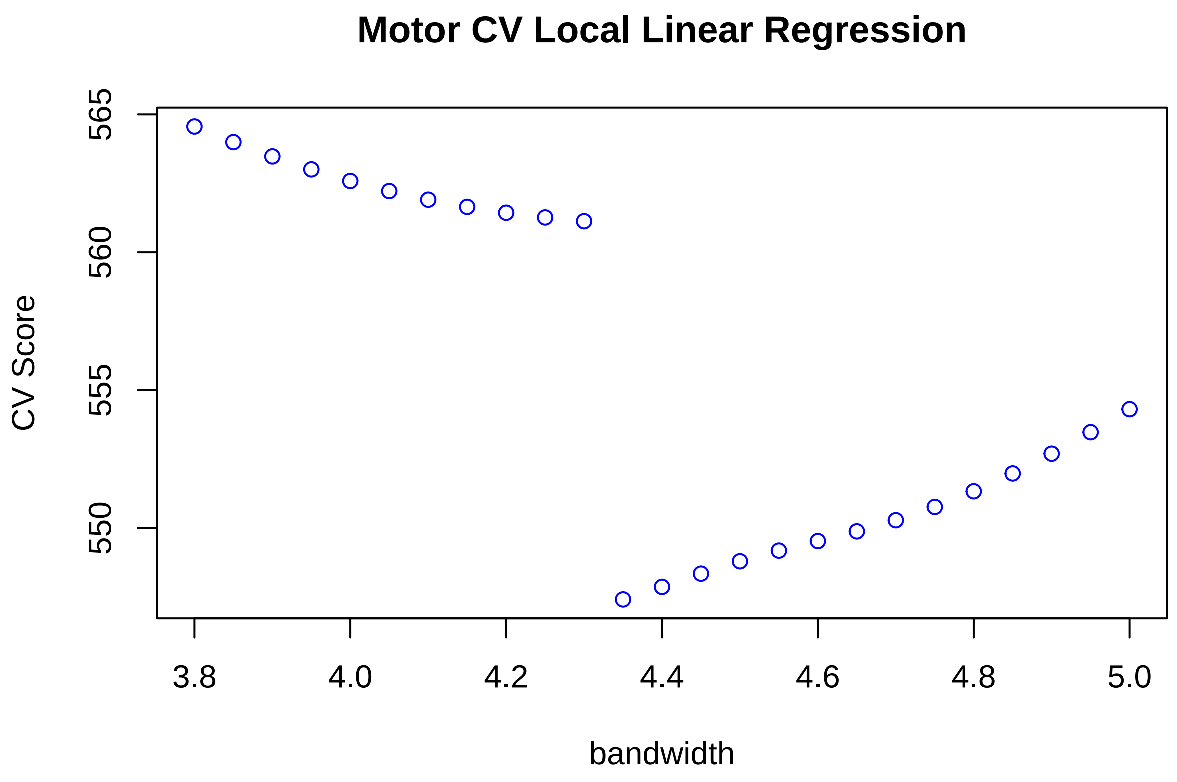
a) Local Linear Smoothing via Local Linear Regression

```
motor <- read.csv("motor.csv") #set the wd!
y <- motor$accel
x <- motor$times
```

```r
#install.packages("locfit")
library(locfit)
```

```
## locfit 1.5-9.1    2013-03-22
```

```r
loocv_reg <- function(y, x, h, kern = "tricube") {
  n <- length(y)
  fit <- locfit(y ~ x, alpha = c(0, h), maxk = 1E5, deg = 1, kern = kern)
  Lii <- predict(fit, where="data", what="infl")
  r_hat <- predict(fit, x)
  term <- (y - r_hat) / (1 - Lii)
  return(sum(term^2) / n)
}
best_h <- function(cv_scores, h) {
  cv_min <- min(cv_scores)
  h_min_index <- match(cv_min, cv_scores)
  h_min <- h[h_min_index]
  return(h_min)
}

h <- seq(3.8, 5, 0.05)
loocv_vals <- c()
for(i in h) {
  loocv_vals <- c(loocv_vals, loocv_reg(y, x, i))
}
plot(loocv_vals ~ h, main = "Motor CV Local Linear Regression", xlab = "bandwidth", ylab = "CV Score", 
```



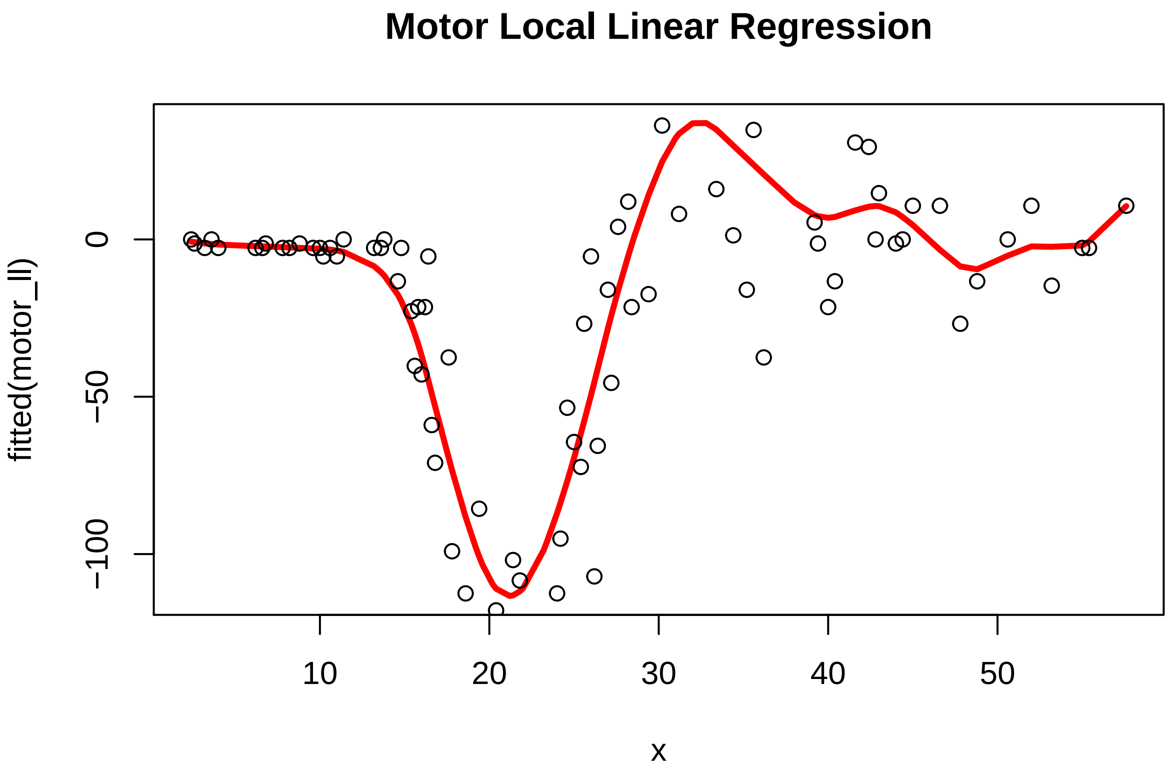
```r
print(cv_min <- min(loocv_vals))
```

```
## [1] 547.4132
```

```
h_min_index <- match(cv_min, loocv_vals)
print(h_min <- h[h_min_index]) #optimal bandwidth
```

```
## [1] 4.35
```

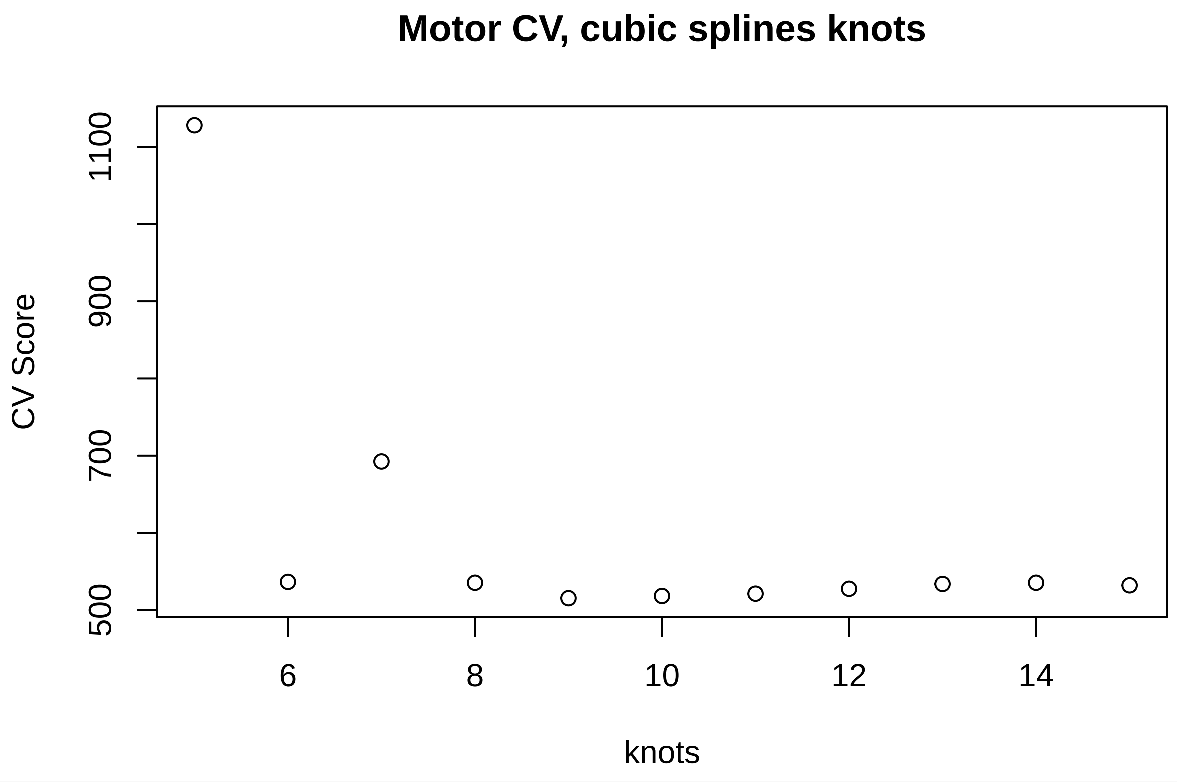
```
motor_ll <- locfit(y ~ x, alpha = c(0, h_min), maxk = 1E5, deg = 1)
plot(fitted(motor_ll) ~ x, type = "l", main = "Motor Local Linear Regression", col = "red", lwd = 3)
points(y ~ x)
```



b) Cubic Regression Spline, Knots equally spaced

```
library(splines)
cubic_spline_cv <- function(y, x, n_knots) {
  errors <- c()
  for(i in 1:length(x)){
    x_train <- x[-i]
    x_test <- x[i]
    y_train <- y[-i]
    y_test <- y[i]
    fit_cv <- smooth.spline(x_train, y_train, nknots = n_knots)
    y_hat <- predict(fit_cv, x_test)
    e <- y_test - y_hat$y
    errors <- c(errors, e^2)
  }
  return(mean(errors))
}
knots <- seq(5, 15, 1)
loocv_vals <- c()
```

```
for(n_knots in knots){
    loocv_vals <- c(loocv_vals, cubic_spline_cv(y, x, n_knots))
}
plot(loocv_vals ~ knots, main = "Motor CV, cubic splines knots", ylab = "CV Score")
```



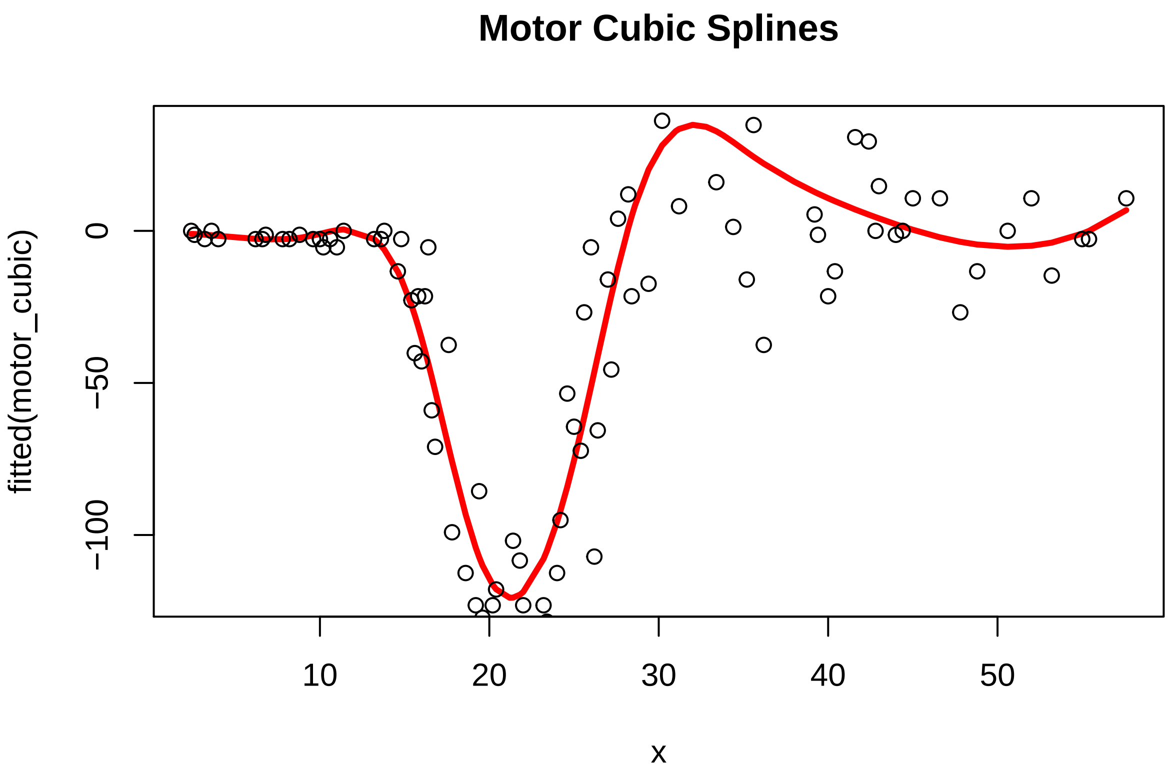
```
print(cv_min <- min(loocv_vals) ) #lowest score knot
```

```
## [1] 515.4223
```

```
knot_min_index <- match(cv_min, loocv_vals)
print(knot_min <- knots[knot_min_index]) #optimal knot
```

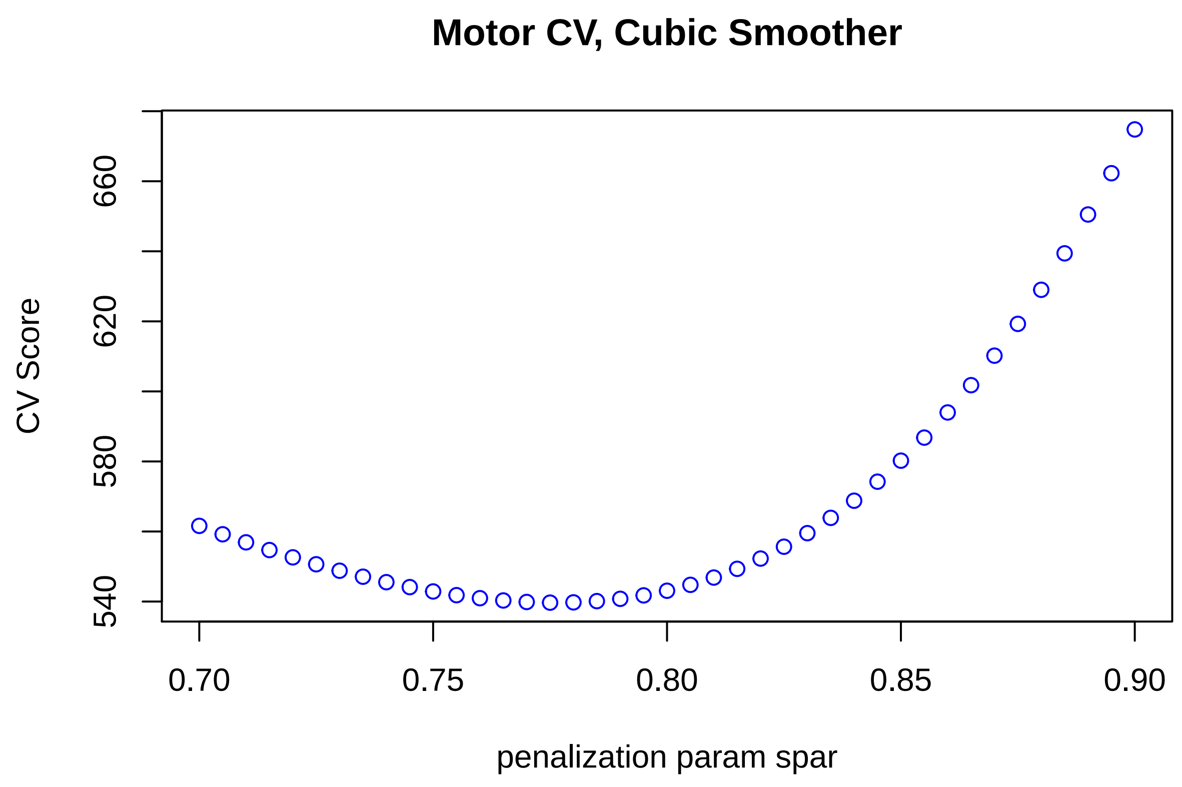
```
## [1] 9
```

```
motor_cubic <- smooth.spline(x, y, nknots = knot_min)
plot(fitted(motor_cubic) ~ x, type = "l", main = "Motor Cubic Splines", col = "red", lwd = 3)
points(y ~ x)
```



c) Knots at data points

```r
spline_smooth_cv <- function(y, x, spar) {
  fit <- smooth.spline(x, y, all.knots = TRUE, spar=lambda)
  Lii <- fit$lev
  r_hat <- fit$y
  term <- (y - r_hat) / (1 - Lii)
  return(mean(term^2))
}

lambdas <- seq(0.7, 0.9, 0.005)
loocv_vals <- c()
for(lambda in lambdas) {
  loocv_vals <- c(loocv_vals, spline_smooth_cv(y, x, lambda))
}
plot(loocv_vals ~ lambdas, main = "Motor CV, Cubic Smoother", xlab = "penalization param spar", ylab =
```



```r
print(cv_min <- min(loocv_vals) )
```

```
## [1] 539.6881
```

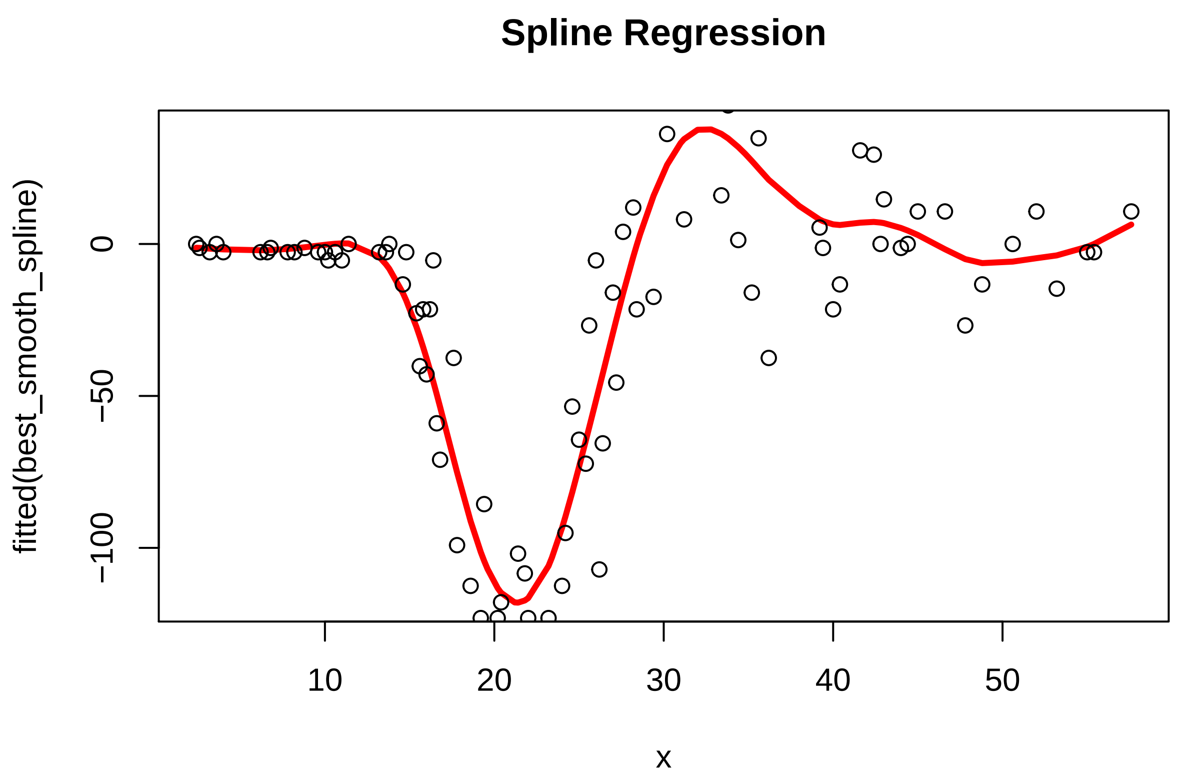
```r
spar_min_index <- match(cv_min, loocv_vals)
print(spar_min <- lambdas[spar_min_index]) #optimal spar
```

```
## [1] 0.775
```

```r
best_smooth_spline <- smooth.spline(x, y, all.knots = TRUE, spar = spar_min)
best_smooth_spline$lambda #best lambda
```

```
## [1] 8.856782e-05
```

```r
plot(fitted(best_smooth_spline) ~ x, type = "l", main = "Spline Regression", col = "red", lwd = 3)
points(y ~ x)
```



comparison)

After adjusting our tuning parameters, Local Linear Regression gives a CV score of 547.8686 (h = 4.35), Cubic with equally spaced knots gives 513.2399 (9 knots), and smoother splines gives 539.6881 (lambda = 8.856782e-05). So the cubic splines works out the best, followed by spline regression, and finally local linear regression.

4) Doppler Effect

a)

```
#install.packages("wavethresh")
library(wavethresh)
```

```
## Loading required package: MASS

## WaveThresh: R wavelet software, release 4.6.8, installed

## Copyright Guy Nason and others 1993-2016

## Note: nlevels has been renamed to nlevelsWT
```

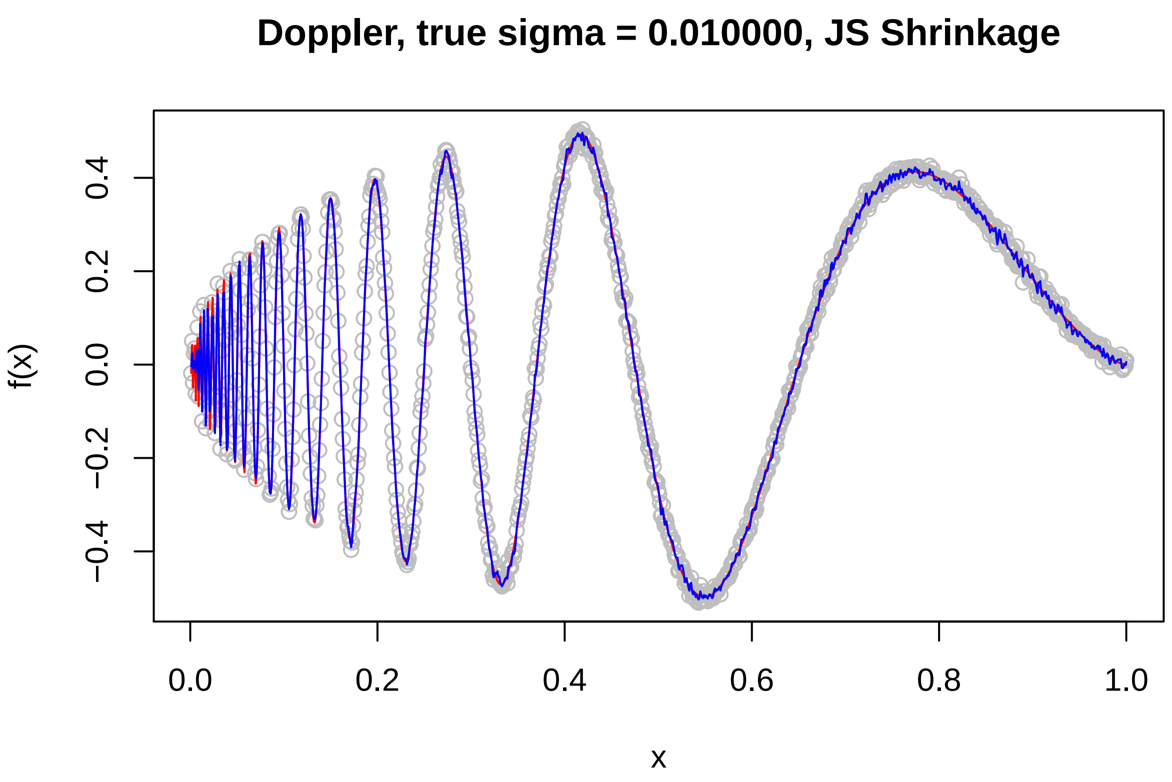
```
doppler <- function(x) {sqrt(x*(1-x)) * sin(2.1*pi/(x+.05))}
n <- 1024
x <- seq(1/n, 1, length.out = n)
```

```r
sigma_1 = 0.01
sigma_2 = 0.1
Y_1 <- doppler(x) + sigma_1 * rnorm(n)
Y_2 <- doppler(x) + sigma_2 * rnorm(n)
```

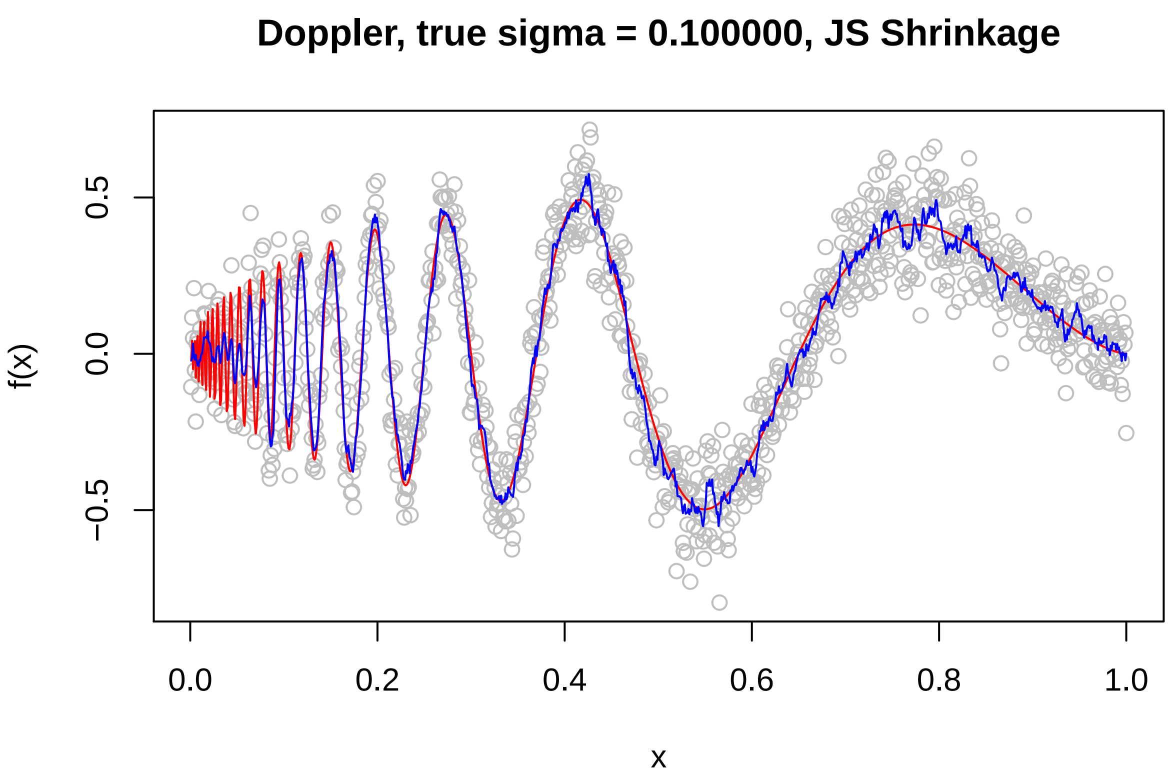
i) James-Stein

$\sigma = 0.01$)

```r
get_js_coeff <- function(n, sigma, shrink){
  return(1 - (n - 2)*sigma^2 / sum(shrink^2))
}

#returns the shrinked JS estimated means with shrinkage around v
shrinkage_js <- function(Z, sigma, v = 0){
  n <- length(Z)
  shrink <- Z - v
  coeff <- get_js_coeff(n, sigma, shrink)
  return(v + coeff * shrink)
}

plot_shrink_js <- function(Y, doppler, sigma, plot_b = FALSE){
  wd_map <- wd(Y, family="DaubLeAsymm", filter.number=8)
  levels <- 0:(wd_map$nlevels - 1)
  coeffs <- c()
  for(l in levels){
    Z <- accessD(wd_map, l)
    est_sigma <- mad(Z)
    coeffs <- c(shrinkage_js(Z, sigma), coeffs)
  }
  wd_map$D <- coeffs
  y_hat <- wr(wd_map)

  title = sprintf("Doppler, true sigma = %2f, JS Shrinkage", sigma)
  if(plot_b){title = sprintf("Doppler Outliers, true sigma = %2f, JS Shrinkage", sigma) }
  plot(Y ~ x, lwd = 1, col = "grey", ylab = "f(x)", main = title)
  curve(doppler, type = "l", col = "red", lwd = 1, add = TRUE, n = 1000)
  lines(y_hat ~ x, type = "l", col = "blue", lwd = 1)
  return(y_hat)
}
js_yhat_1 = plot_shrink_js(Y_1, doppler, sigma_1)
```



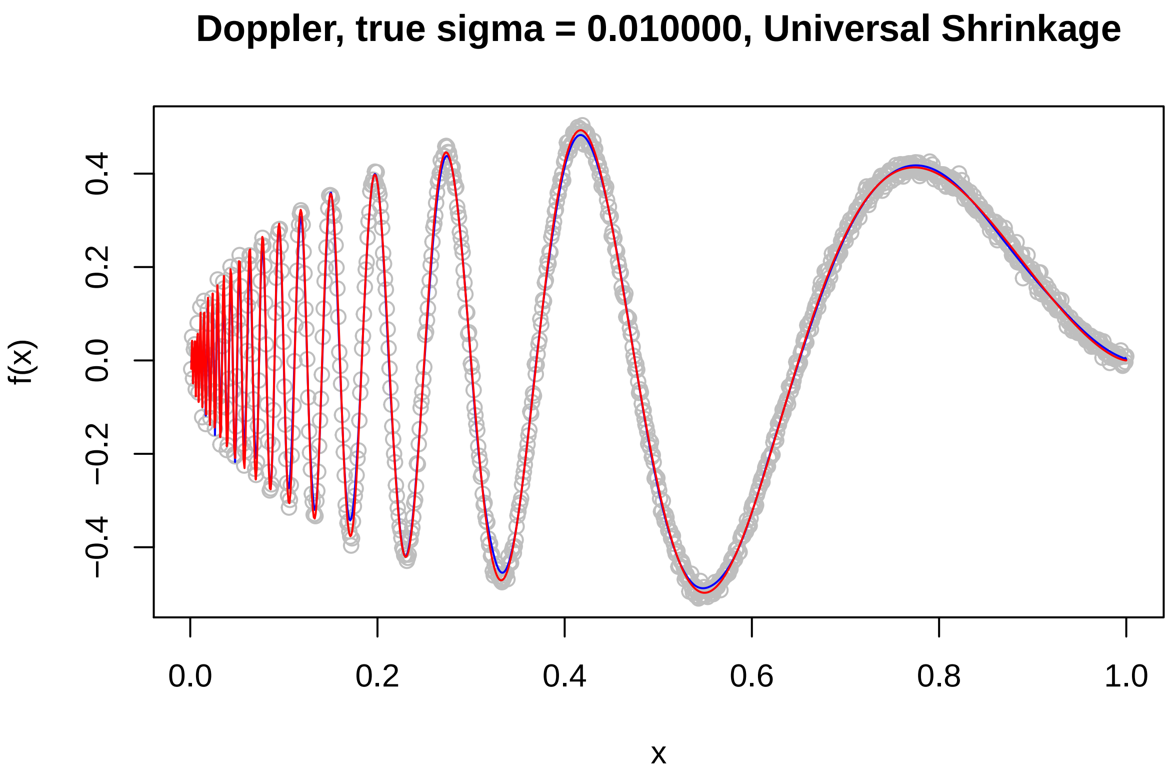
```r
js_yhat_2 = plot_shrink_js(Y_2, doppler, sigma_2)
```



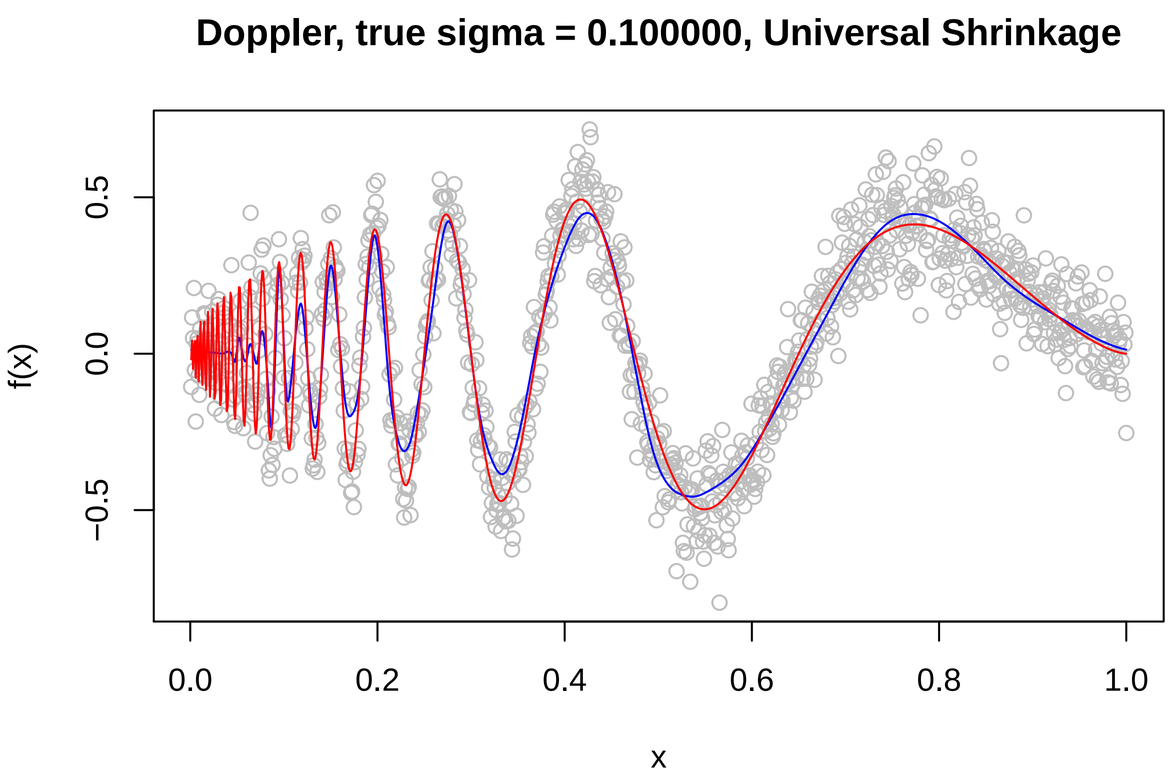
The red line is the true doppler, and blue is the estimated function using james stein shrinkage at each level.

ii) Universal Shrinkage

```r
plot_universal_wavelet <- function(Y, sigma, plot_b = FALSE){
  wd_map <- wd(Y, family="DaubLeAsymm", filter.number=8)
  levels <- 0:(wd_map$nlevels - 1)
  #sigma <- mad(accessD(wd_map, wd_map$nlevels - 1) )
  softthreshwmap <- threshold(wd_map, type="soft", policy="universal")
  y_hat <- wr(softthreshwmap)
  title <- sprintf("Doppler, true sigma = %2f, Universal Shrinkage", sigma)
  if(plot_b){sprintf("Doppler Outliers, true sigma = %2f, Universal Shrinkage", sigma)}

  plot(Y ~ x, lwd = 1, col = "grey", ylab = "f(x)", main = title)
  lines(y_hat ~ x, type = "l", col = "blue", lwd = 1, ylab = "f(x)", main = title)
  curve(doppler, type = "l", col = "red", lwd = 1, add = TRUE, n = 1000)
  return(y_hat)
}
u_s_1 = plot_universal_wavelet(Y_1, sigma_1)
```



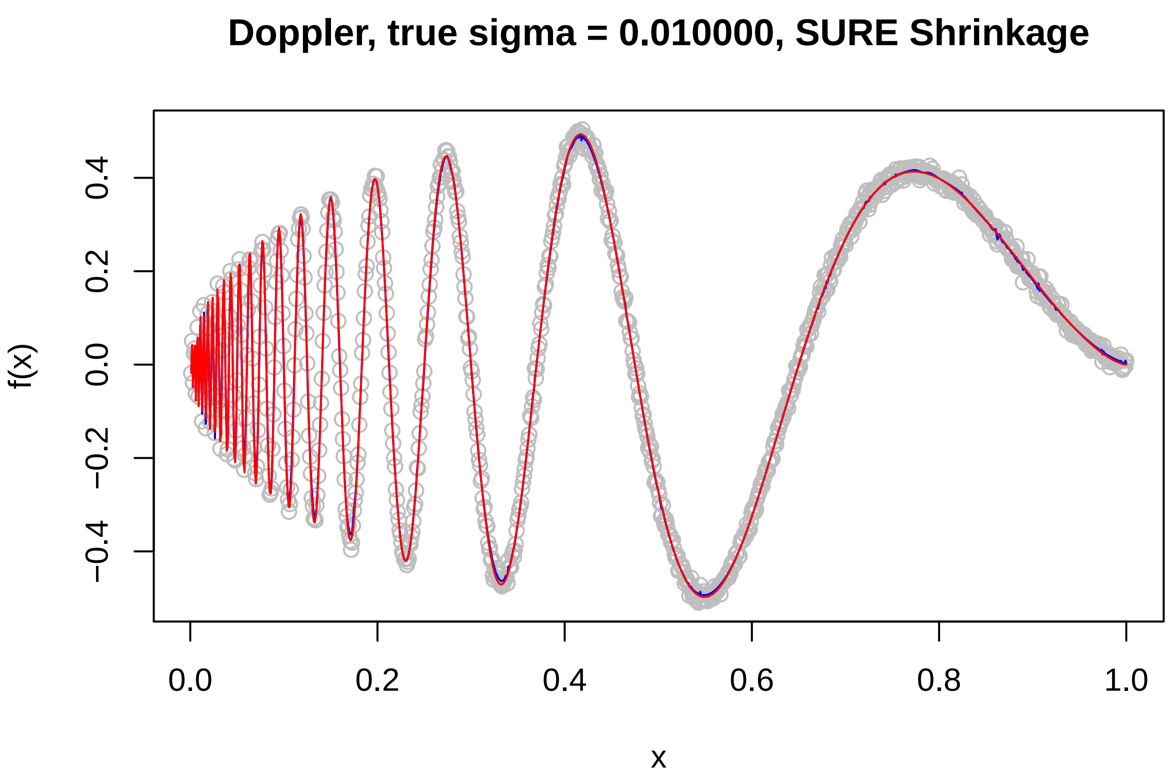
```r
u_s_2 = plot_universal_wavelet(Y_2, sigma_2)
```



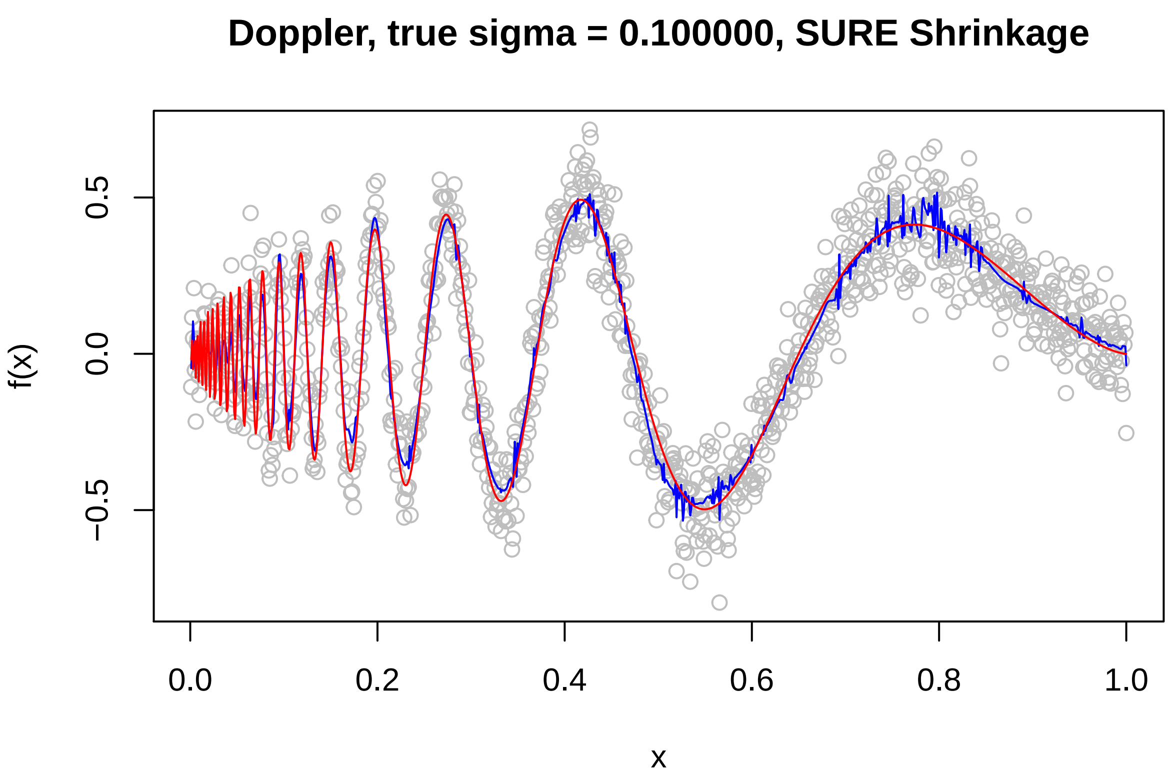
The red line is the true doppler, and blue is the estimated function using universal shrinkage.

iii) Shrinkage to minimize SURE

```
plot_sure_wavelet <- function(Y, sigma, plot_b = FALSE){
  wd_map <- wd(Y, family="DaubLeAsymm", filter.number=8)
  #sigma <- mad(accessD(wd_map, wd_map$nlevels - 1) )
  softthreshwmap = threshold(wd_map, type="soft", policy="sure")
  yhat = wr(softthreshwmap)
  title = sprintf("Doppler, true sigma = %2f, SURE Shrinkage", sigma)
  if(plot_b){ title = sprintf("Doppler Outliers, true sigma = %2f, SURE Shrinkage", sigma)}

  plot(Y ~ x, lwd = 1, col = "grey", ylab = "f(x)", main = title)
  lines(yhat ~ x, type = "l", col = "blue", lwd = 1, ylab = "f(x)", main = title)
  curve(doppler, type = "l", col = "red", lwd = 1, add = TRUE, n = 1000)
  return(yhat)
}
sure_1 = plot_sure_wavelet(Y_1, sigma_1)
```



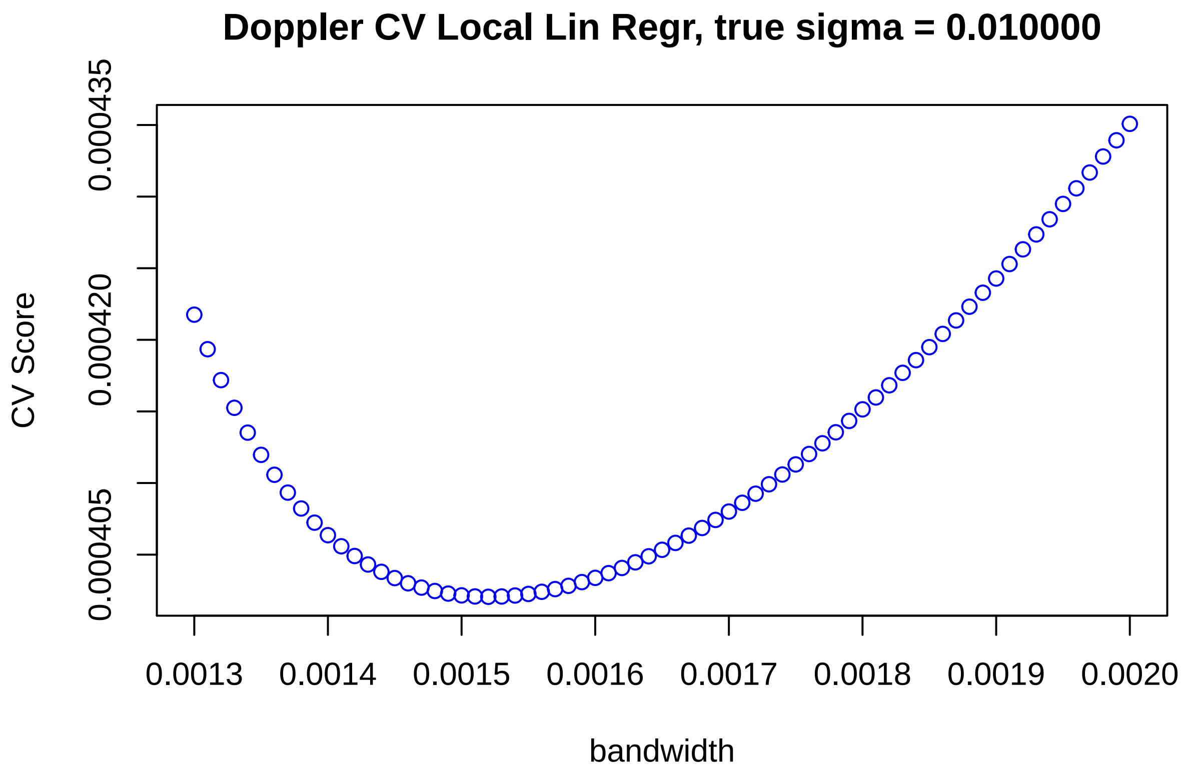
```
sure_2 = plot_sure_wavelet(Y_2, sigma_2)
```



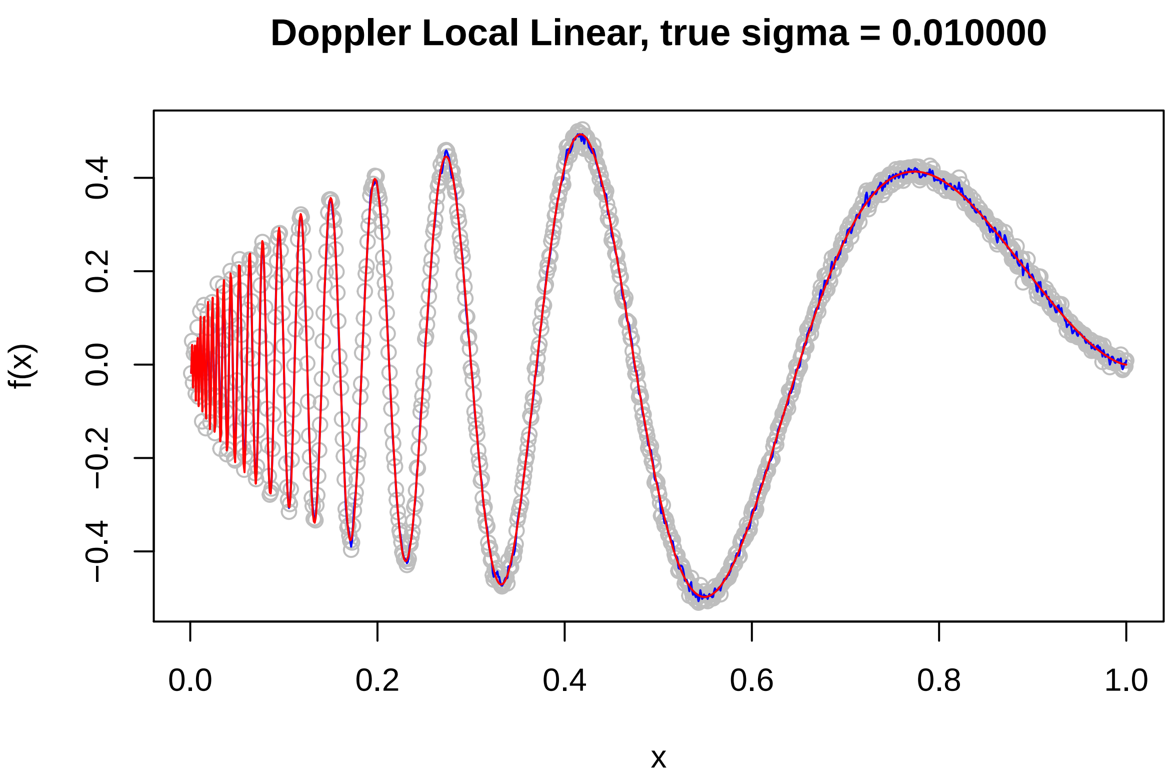
The red line is the true doppler, and blue is the estimated function minimizing sure . ### local linear regression)

```
plot_local_linear <- function(y, x, h, sigma, plot_b = FALSE){
  loocv_vals <- c()
  for(i in h) {
```

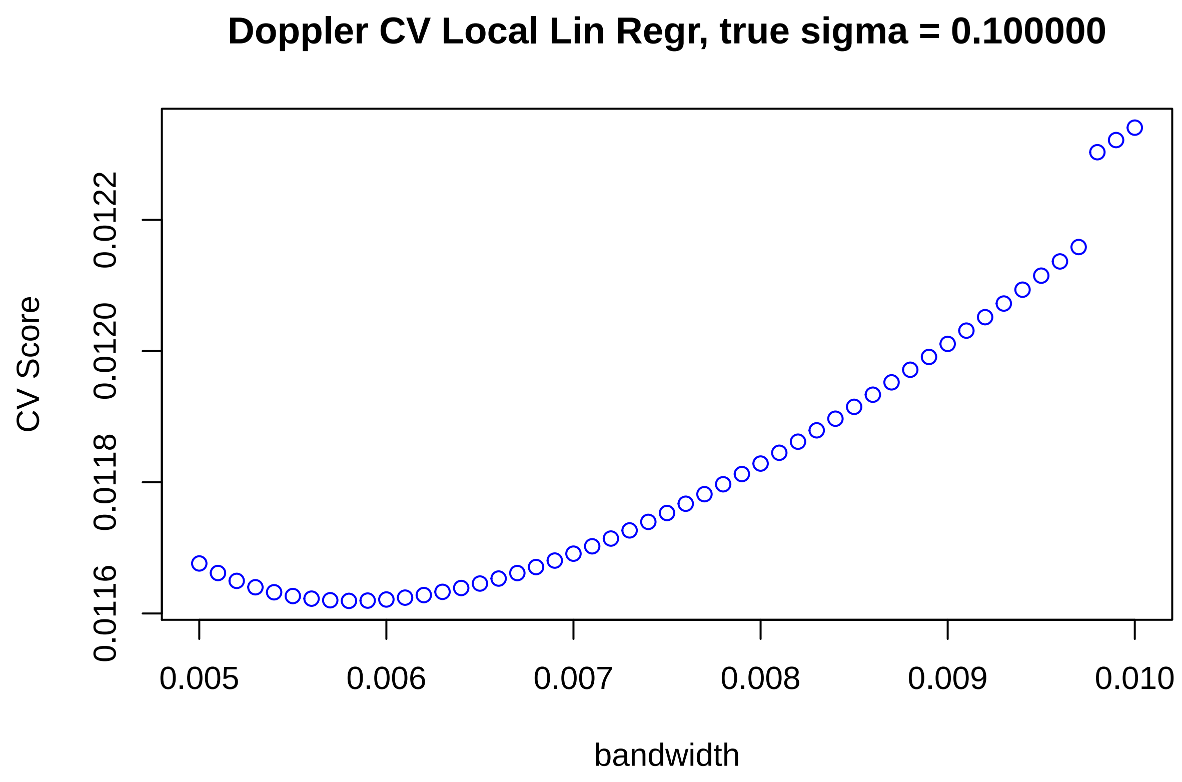
```r
    loocv_vals <- c(loocv_vals, loocv_reg(y, x, i, kern = "gaussian"))
  }

  title_CV = sprintf("Doppler CV Local Lin Regr, true sigma = %2f", sigma)
  if(plot_b){title_CV = sprintf("Doppler Outliers CV Local Lin Regr, true sigma = %2f", sigma)}
  plot(loocv_vals ~ h, main = title_CV, xlab = "bandwidth", ylab = "CV Score", col = "blue")
  print(cv_min <- min(na.omit(loocv_vals))) #min cv
  h_min_index <- match(cv_min, loocv_vals)
  print(h_min <- h[h_min_index]) #optimal bandwidth

  title_doppler = sprintf("Doppler Local Linear, true sigma = %2f", sigma)
  if(plot_b){title_doppler = sprintf("Doppler Outlier Local Linear, true sigma = %2f", sigma)}
  llr_fit <- locfit(y ~ x, alpha = c(0, h_min), maxk = 69000, deg = 1, kern = "guassian")

  plot(y ~ x, lwd = 1, col = "grey", ylab = "f(x)", main = title_doppler)
  lines(fitted(llr_fit) ~ x, type = "l", col = "blue", lwd = 1,
        xlab = "x", ylab = "f(x)", main = title_doppler)
  curve(doppler, type = "l", col = "red", lwd = 1, add = TRUE, n = 1000)
}

h_1 <- seq(0.0013, 0.002, 0.00001)
plot_local_linear(Y_1, x, h_1, sigma_1)
```



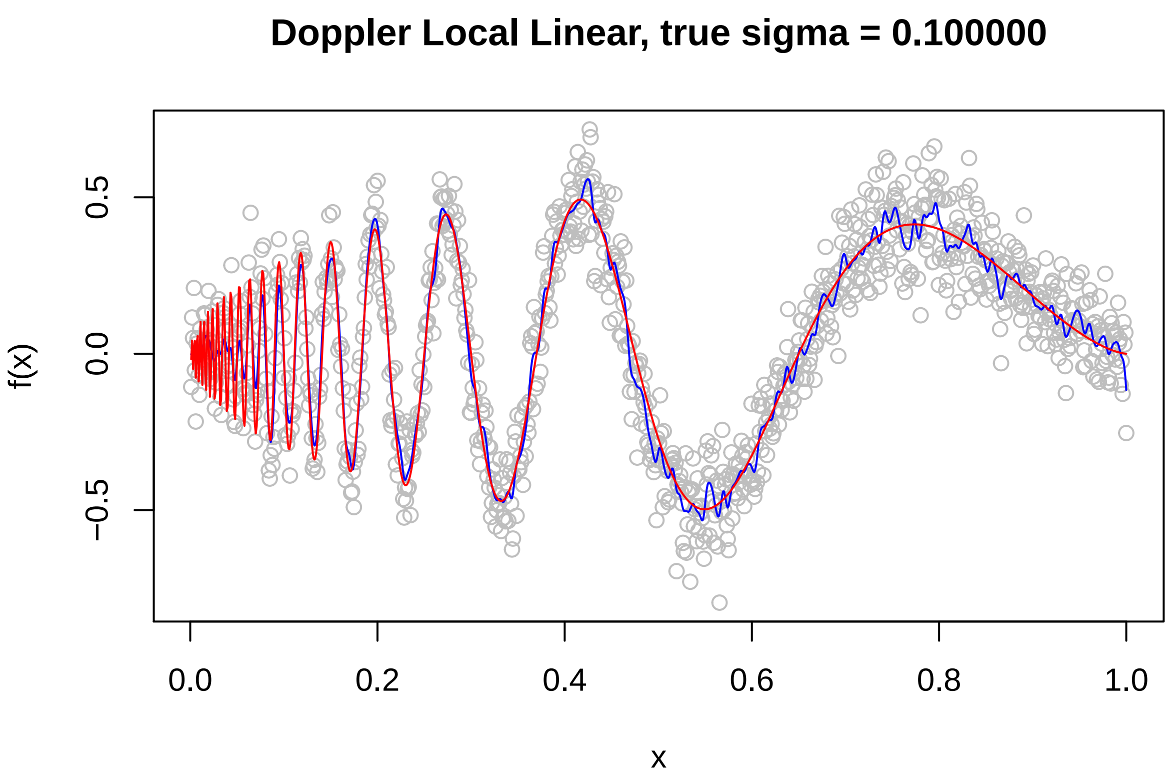
```
## [1] 0.0004020576
## [1] 0.00152
```



```r
h_2 <- seq(0.005, 0.01, 0.0001)
plot_local_linear(Y_2, x, h_2, sigma_2)
```



```
## [1] 0.01161924
## [1] 0.0058
```



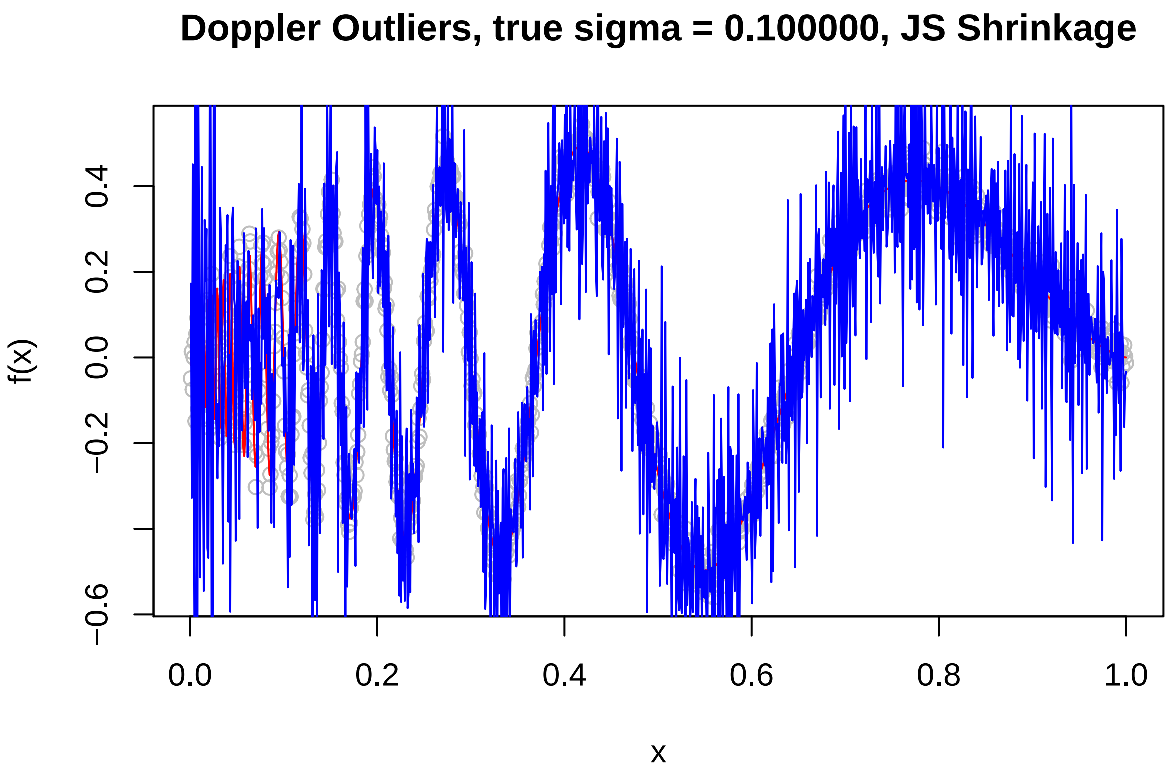
The red is the true doppler, and the blue is the estimated using local linear regression. Sure is similar, but has less variance.

Comparison)

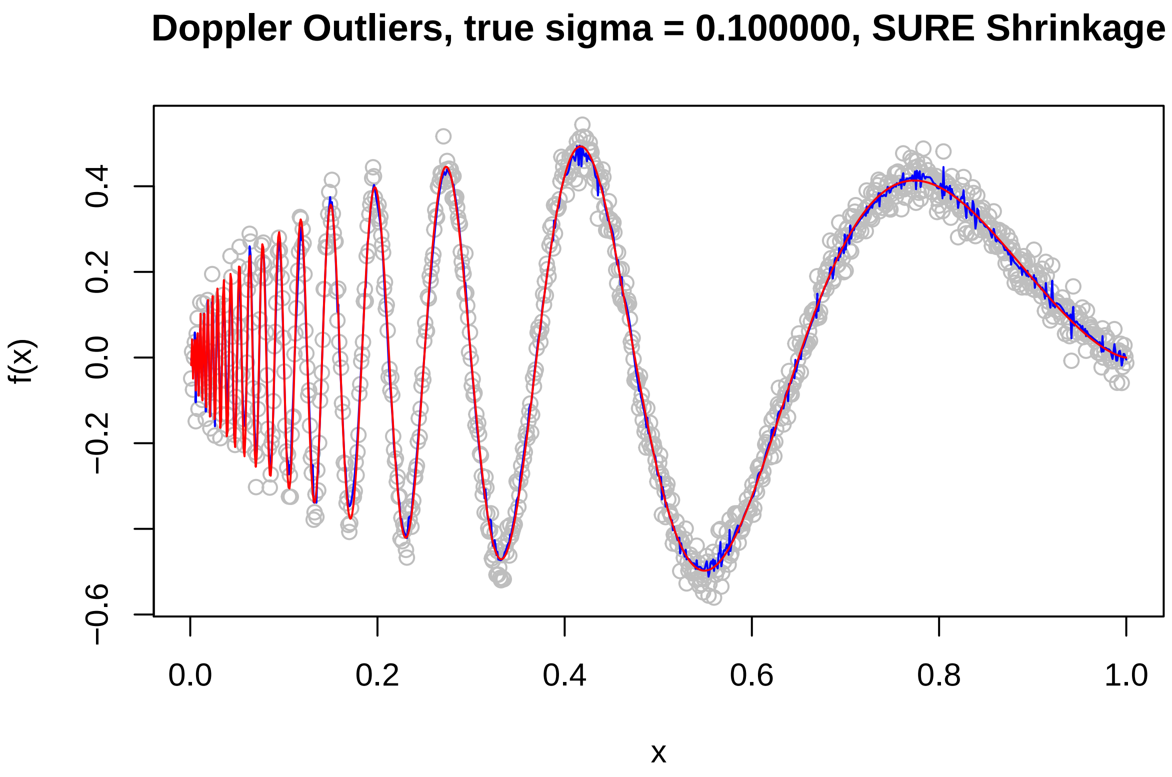
When $\sigma = 0.01$, Local Linear Regression hsd low bias, but is somewhat influenced by the variance in the data points. James Stein shrinkage at all levels does about the same. SURE shrinkage also has low bias and lower variance than both. Universal has low bias as well, and even lower variance than the other three.
When $\sigma = 0.1$, local linear is has very low bias, but the variance is noticably larger than when $\sigma = 0.01$. SURE shrinkage and James Stein does similarly. Universal shrinkage is more biased than all three, but much lower variance.

b Crazy Variance for epsilon)

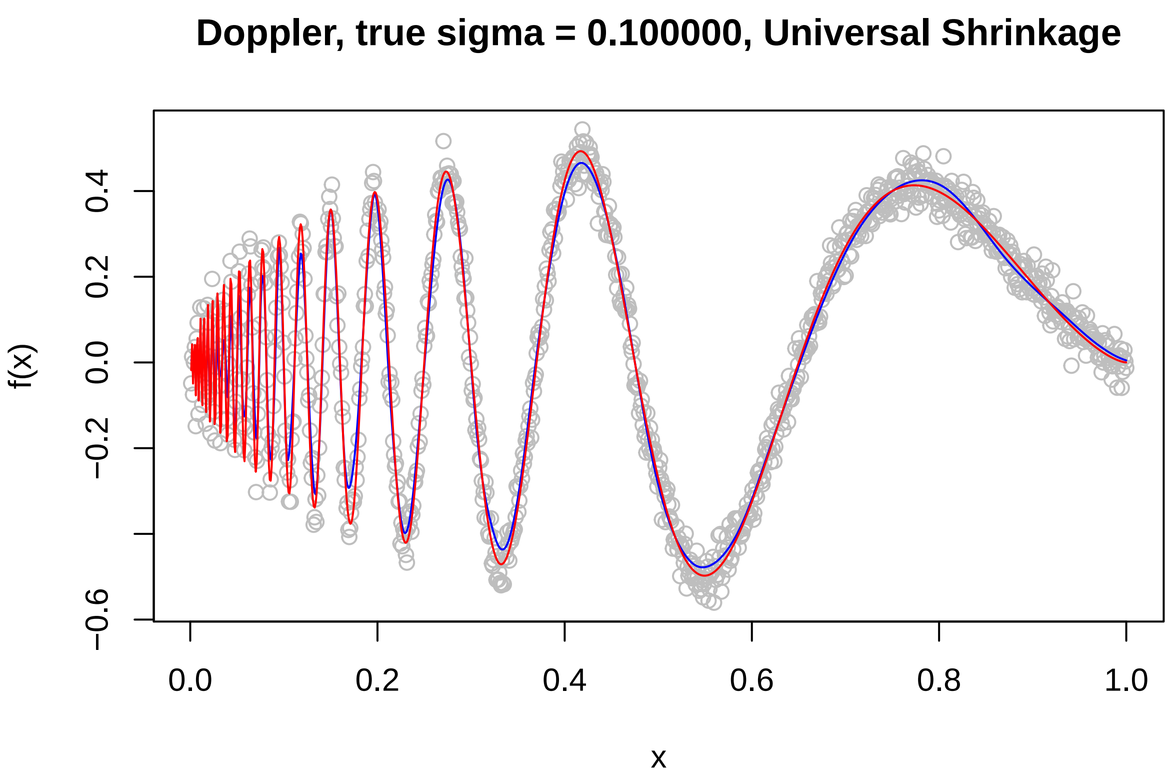
```r
sigma_crazy = sqrt( (19/20)^2 * 0.1 + (1/20)^2 * 4)
e = rnorm(n, sd = sigma_crazy)

#Y_3 <- doppler(x) + sigma_1 * e
Y_4 <- doppler(x) + sigma_2 * e

#js_crazy = plot_shrink_js(Y_3, doppler, sigma_1, plot_b = TRUE)
#sure_crazy = plot_sure_wavelet(Y_3, sigma_1)
#uni_crazy = plot_universal_wavelet(Y_3, sigma_1)
#llr_crazy = plot_local_linear(Y_3, x, seq(0.008, 0.02, 0.0003), sigma_1)

js_crazy = plot_shrink_js(Y_4, doppler, sigma_2, plot_b = TRUE)
```



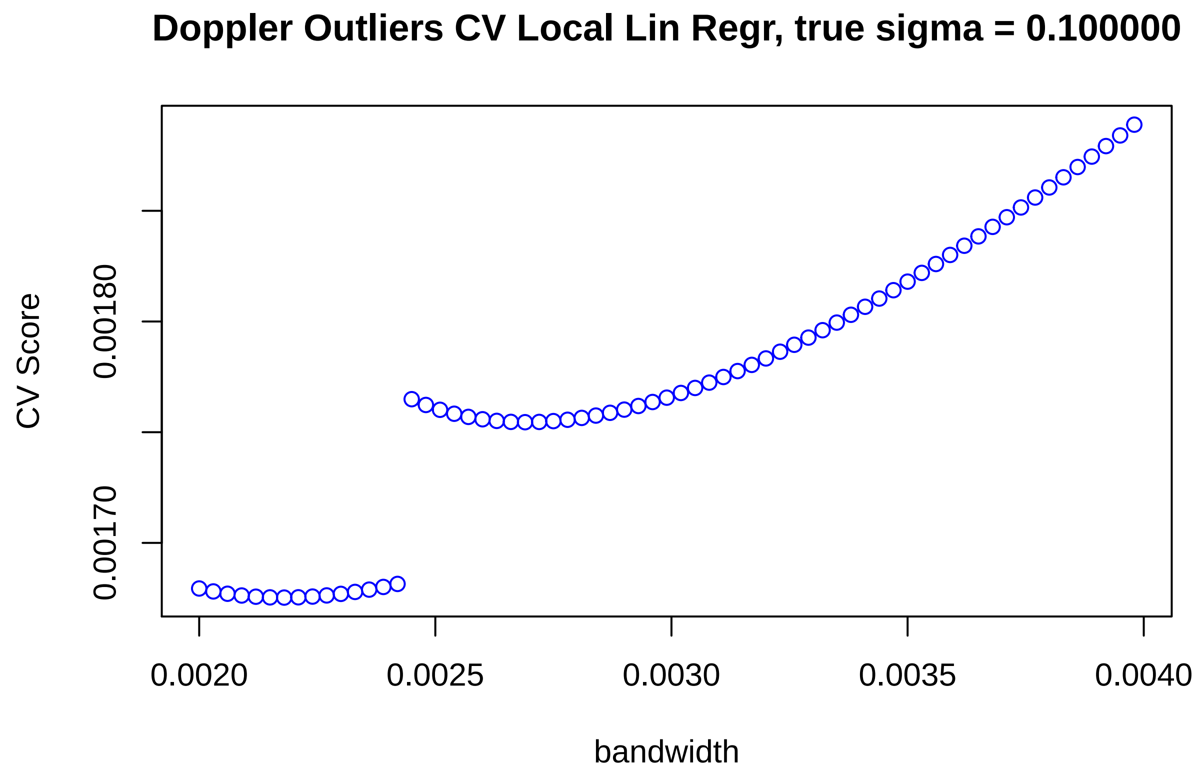
```
sure_crazy = plot_sure_wavelet(Y_4, sigma_2, plot_b = TRUE)
```



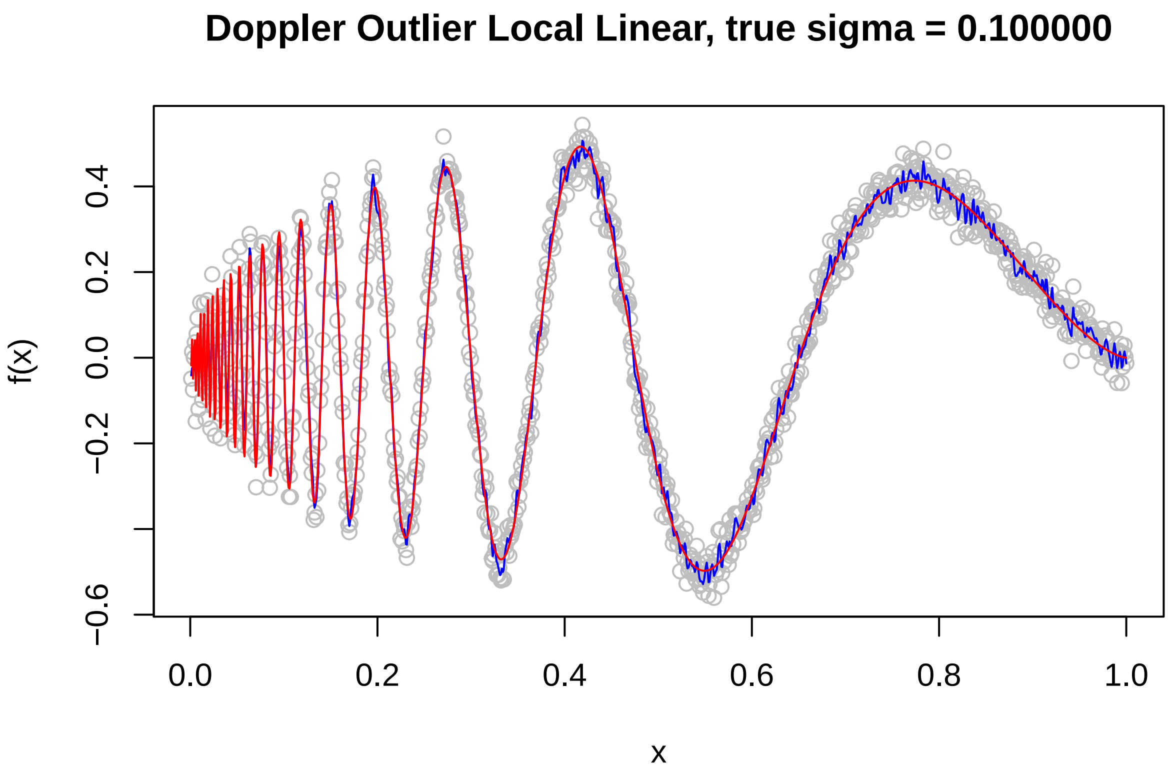
```
uni_crazy = plot_universal_wavelet(Y_4, sigma_2, plot_b = TRUE)
```



```
llr_crazy = plot_local_linear(Y_4, x, seq(0.002, 0.004, 0.00003), sigma_2, plot_b = TRUE)
```



```
## [1] 0.001675295
## [1] 0.00218
```



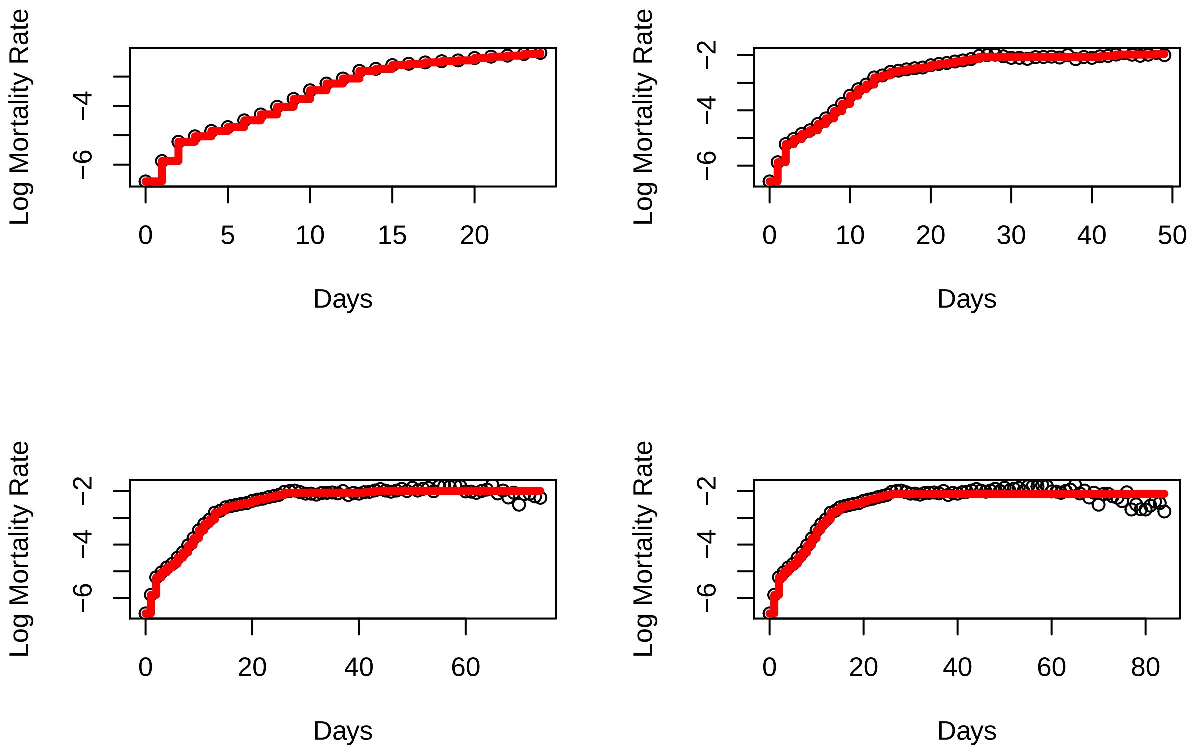
Comparison)

Local Linear Regression has very low bias and high variance. SURE shrink has low bias and variance slightly lower than local linear. James Stein also has low bias, but the variance is higher than both local linear and SURE shrinkage - it clearly suffers from overfitting to the points (you can't even see the points anymore). Universal shrinkage, similar to part a), has nearly no variance but it slightly biased compared to the other three.

5) Fruit Flies

a)

```
flies <- read.csv("flies.dat", sep="")
e <- min(flies$mort.rate[flies$mort.rate > 0])
log_mort <- log(flies$mort.rate + e)
d <- c(25, 50, 75, 85)
par(mfrow = c(2,2))
for(days in d){
  log_mort_d <- log_mort[1:days]
  d_days <- flies$day[1:days]
  iso_est <- pava(log_mort_d)
  plot(log_mort_d ~ d_days, xlab = "Days", ylab = "Log Mortality Rate")
  lines(iso_est ~ d_days, type = "s", lwd = 4, col = "red")
}
```

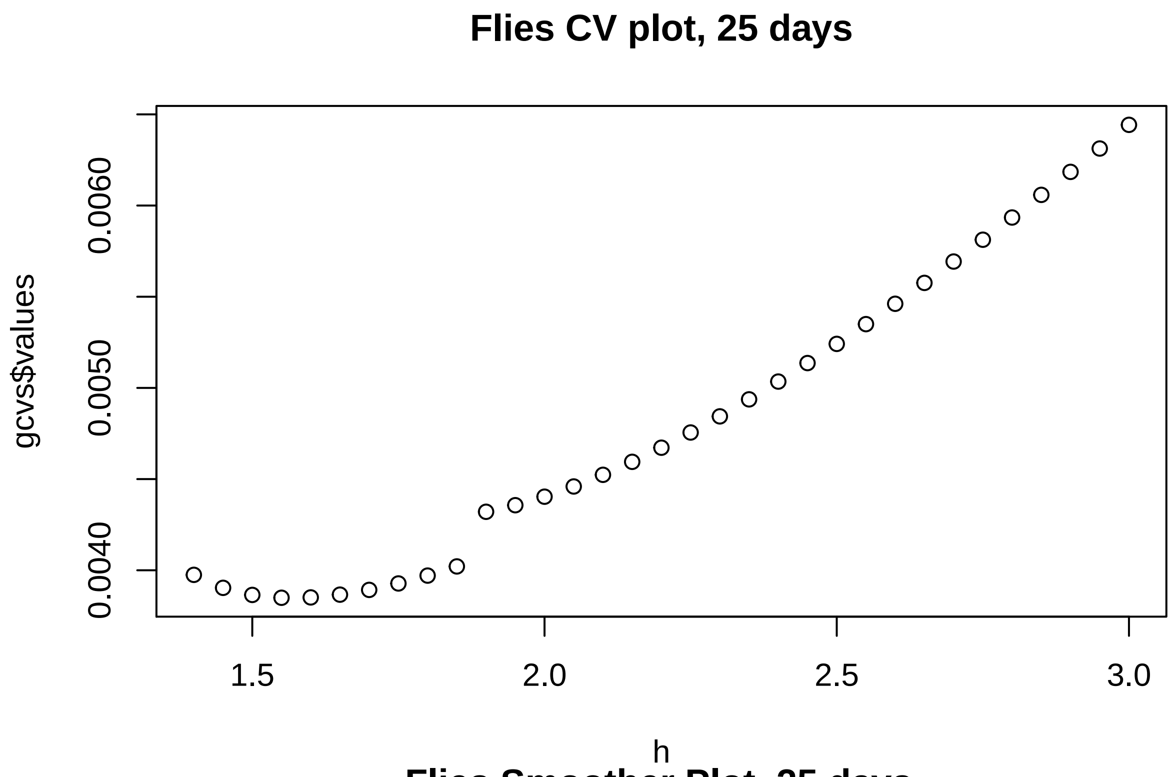


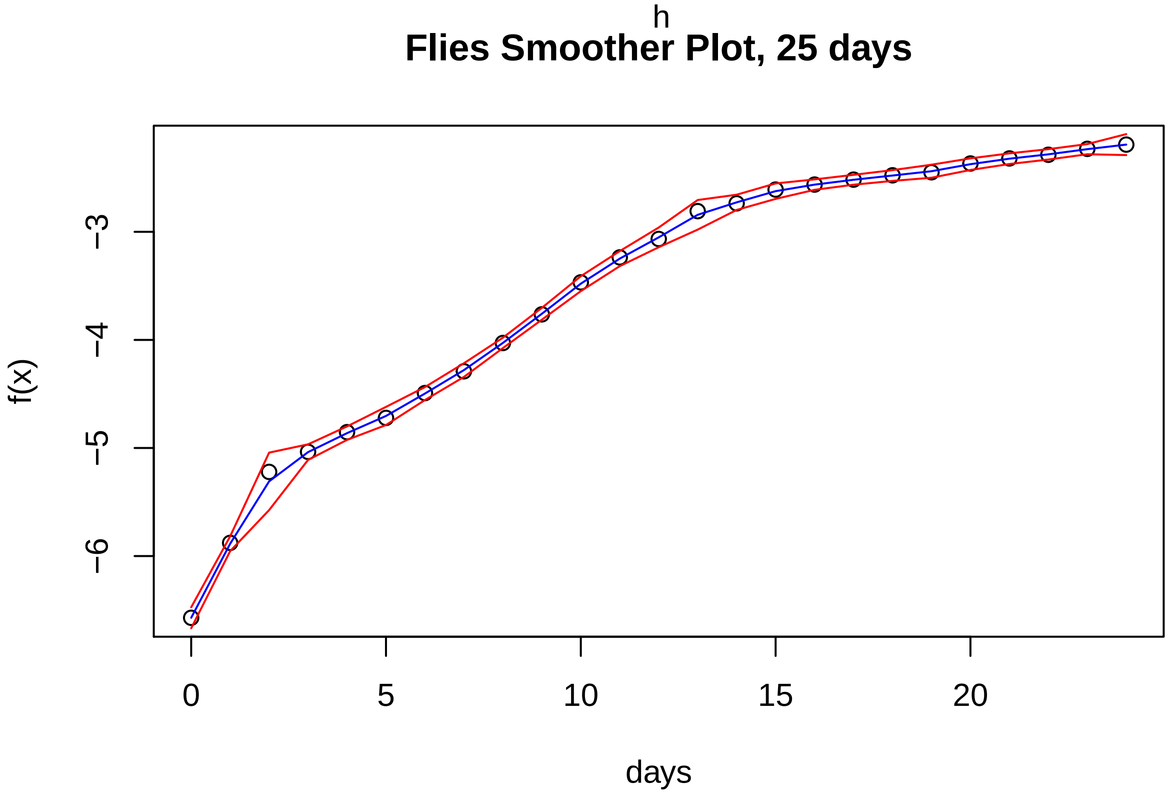
For d = 25, 50, monotonic regression works very well. For most part, the log mortality rates strictly increasing. Even at d = 75, monotonic regression fits quite well, except near the end. At d = 85, however, it is clear that the log mortality rates are no longer strictly nondecreasing.
I propose two statistical ways to determine at which day d the response fails to be monotonic. The first is to check the mean squared error of the sample average of the first d days versus the mean squared error of the isotonic regression. When the MSE of isotonic regression surpasses that of the MLE, then we conclude that the response is not monotonic. A second method would be to look at the number of times PAVA switches data points. Once it passes a certain point, k, then we conclude that the response is not monotonic.

b) Nonparametric Smoother - Local Linear Regression

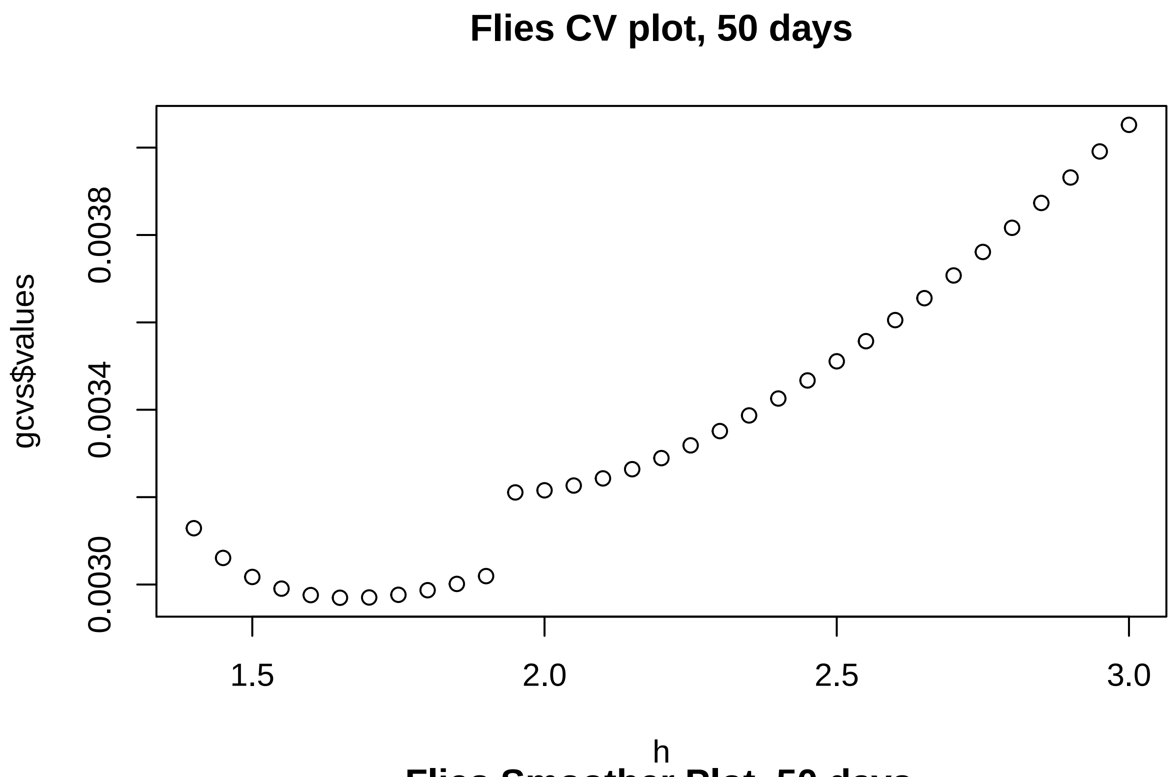
```
#function for estimating sigma in the heterocedastic case, returns
#q(x) where sigma(x) = e^(q(x))
get_q_hetero <- function(y, x, fit, h){
  epsilon = 0.0001
  l2 = (y - fitted(fit))^2 + epsilon
  z = log(l2)
  alphas = cbind(rep(0,length(h)), h)
  gcvs = gcvplot(z ~ x, alpha=alphas, maxk = 100000, deg=1)
  #title = sprintf("Hetero Smoother CV plot")
  #plot(h, gcvs$values, type="p", main = title)
  hstar = h[which(gcvs$values == min(gcvs$values))]
  #print(hstar)
  q = locfit(z ~ x, alpha=c(0,hstar), deg=1, maxk = 100000)
  return(q)
}

plot_smooth_bands <- function(log_mort, days, d, h, h_q){
  y = log_mort[1:d]
```

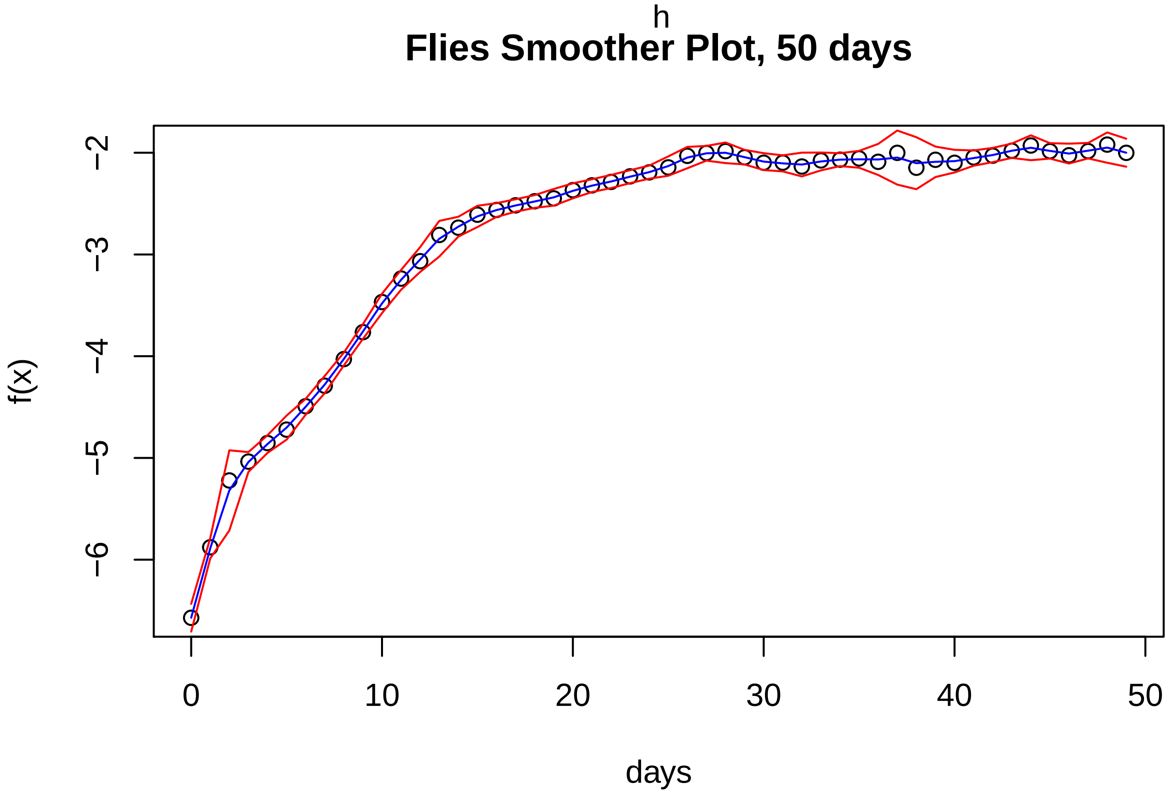
```r
  x = days[1:d]

  alphas = cbind(rep(0,length(h)), h)
  gcvs = gcvplot(y ~ x, alpha=alphas, maxk = 100000, deg=1, kern = "gaussian")
  title = sprintf("Flies CV plot, %d days", d)
  plot(h, gcvs$values, main = title)

  hstar = h[which(gcvs$values == min(gcvs$values))]
  llr_fit = locfit(y ~ x, alpha=c(0,hstar), deg=1, maxk = 69000, kern = "gaussian")

  title = sprintf("Flies Smoother Plot, %d days", d)
  plot(x, y, xlab = "days", ylab = "f(x)", main = title)

  q = get_q_hetero(y, x, llr_fit, h_q)
  sigmas = sqrt( exp(fitted(q)) )

  normell <- predict(llr_fit, where="data", what="vari")
  n = length(x)
  lines(x, fitted(llr_fit), 'l', col = 'blue', lwd = 1)
  lines(x, fitted(llr_fit) + sqrt(n)*1.96*sigmas*normell, 'l', col='red', lwd=1)
  lines(x, fitted(llr_fit) - sqrt(n)*1.96*sigmas*normell, 'l', col='red', lwd=1)
}
par(mfrow = c(1,1))
h_days_1 = seq(1.4, 3, 0.05)
h_days_2 = h_days_1
h_days_3 = seq(2, 4, 0.05)
h_days_4 = seq(2, 5, 0.07)
h_days_5 = seq(7, 20, 0.3)
d[5] = nrow(flies)

plot_smooth_bands(log_mort, flies$day, d[1], h_days_1, h_days_1)
```



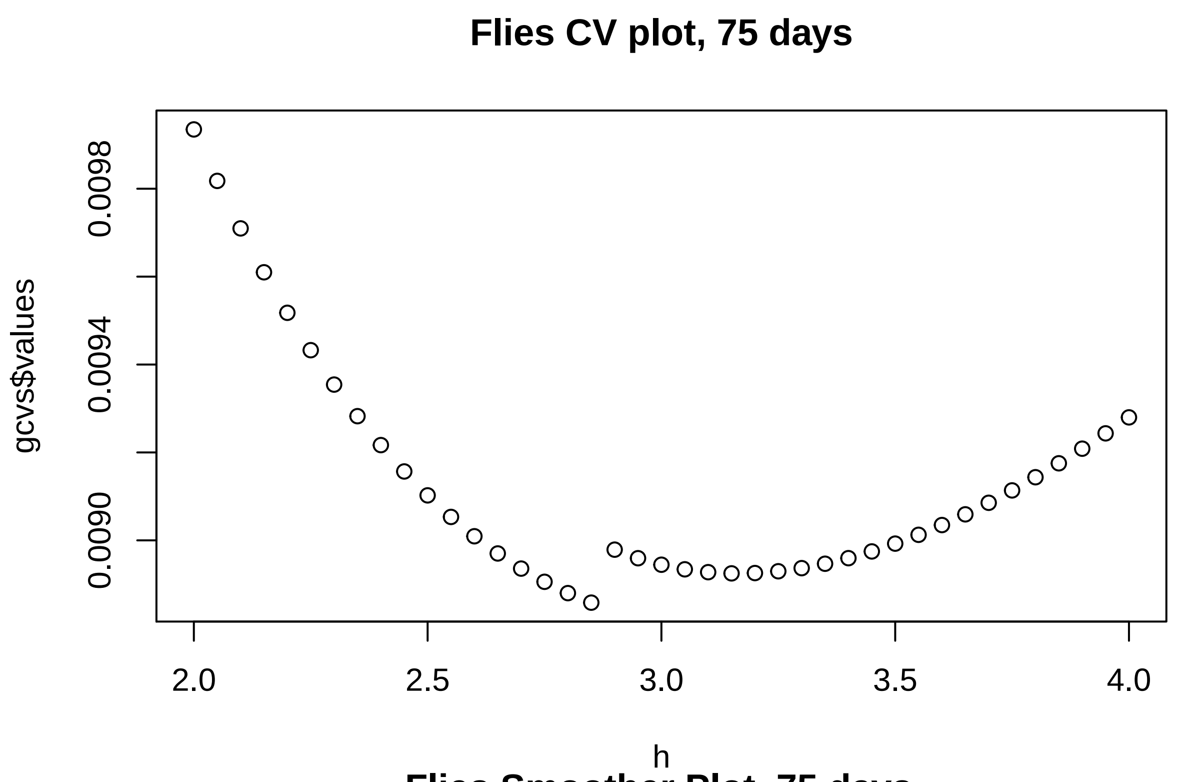


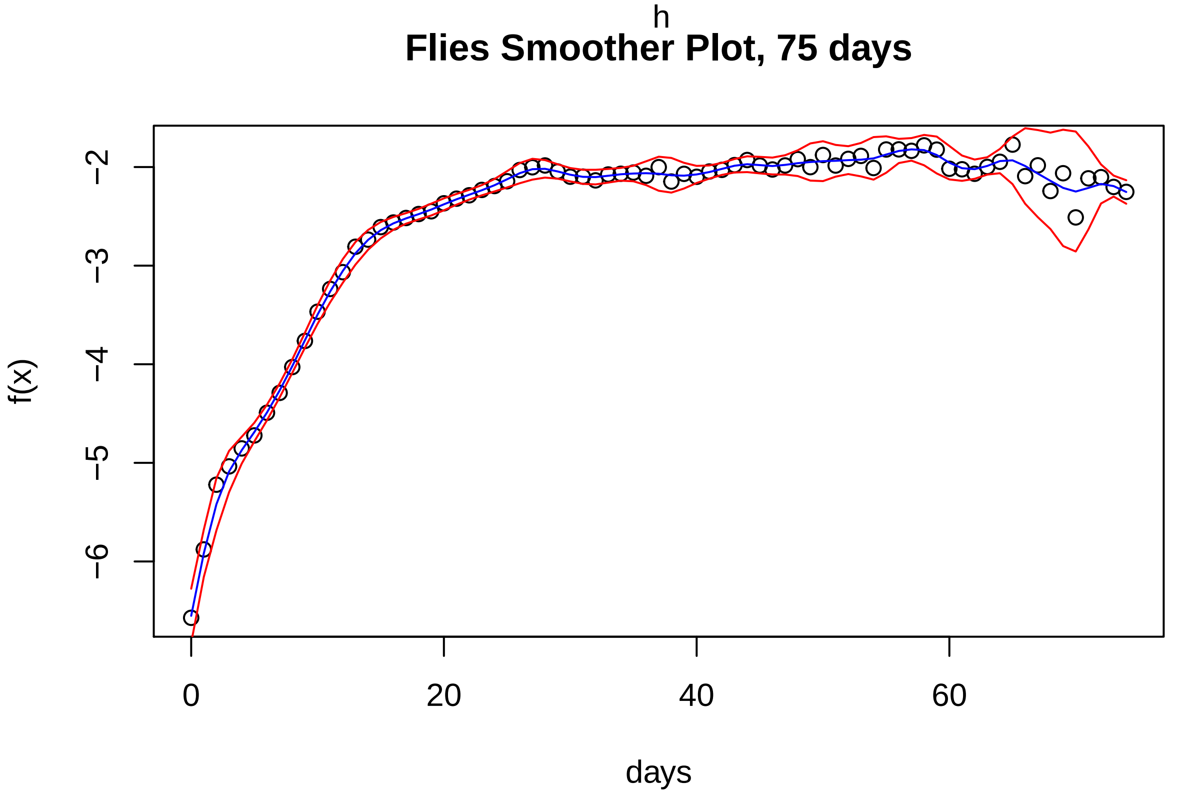
```
plot_smooth_bands(log_mort, flies$day, d[2], h_days_2, h_days_2)
```



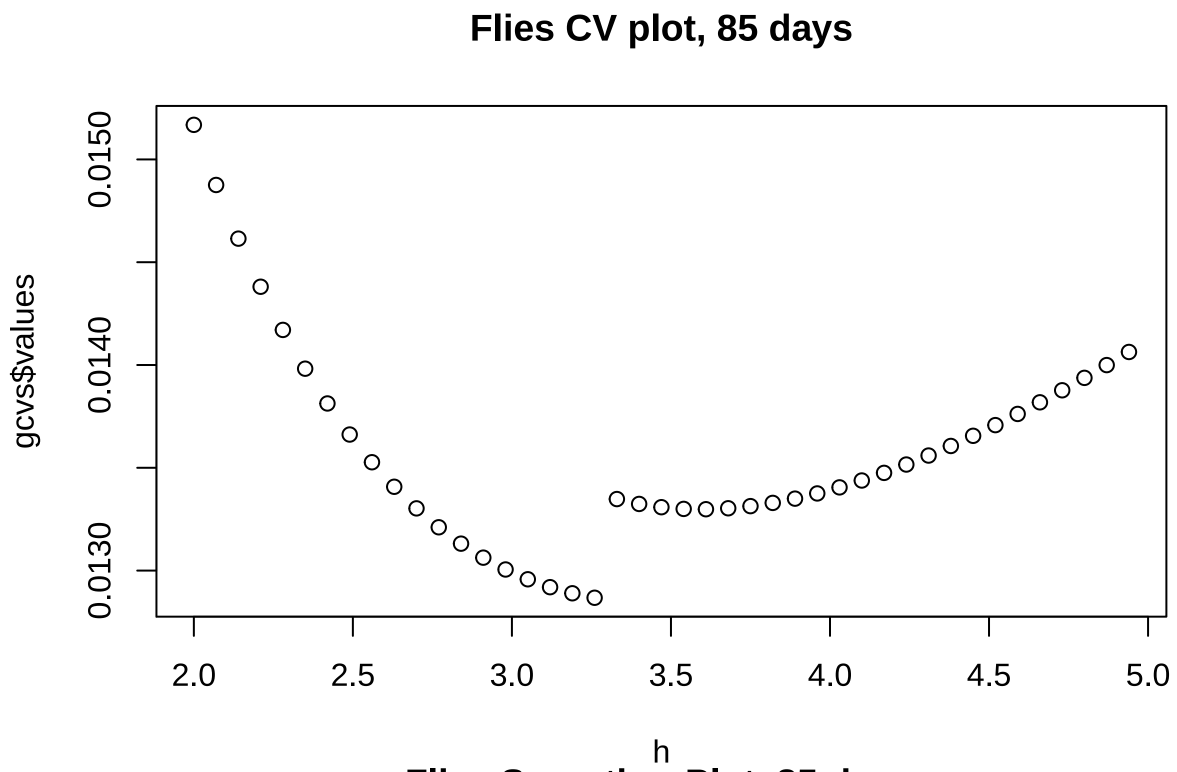


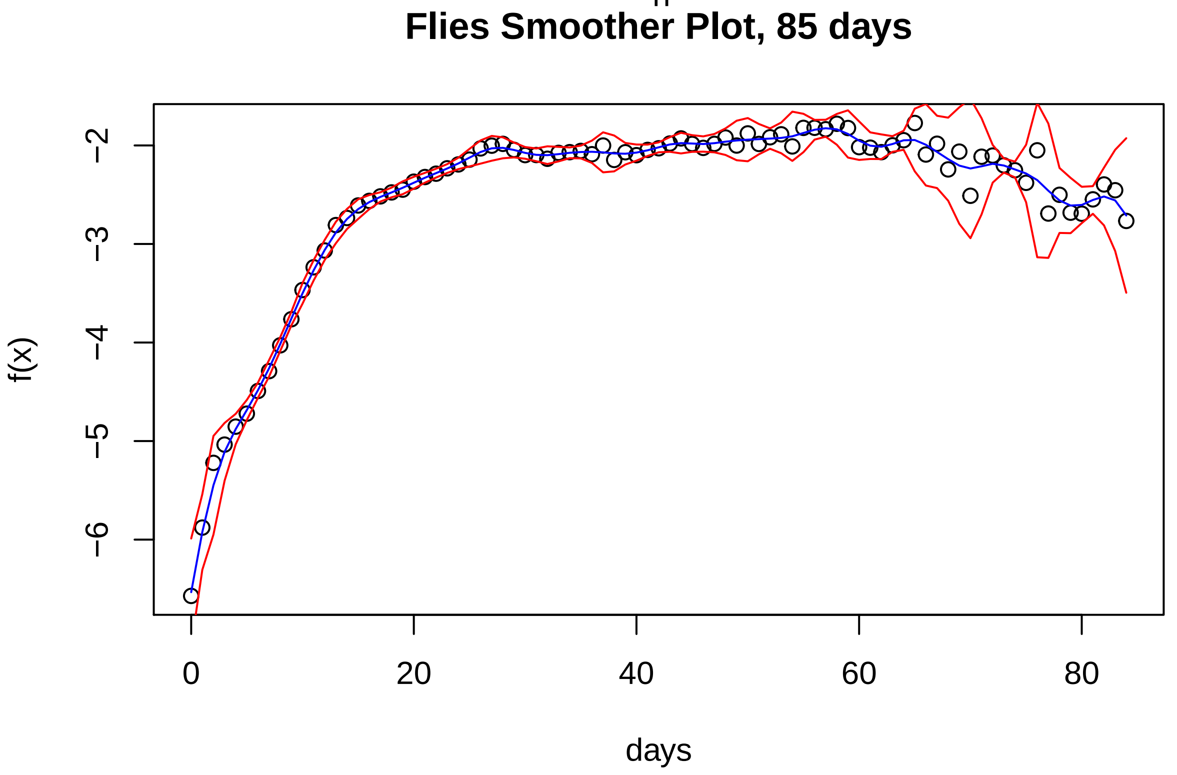
```
plot_smooth_bands(log_mort, flies$day, d[3], h_days_3, h_days_3)
```



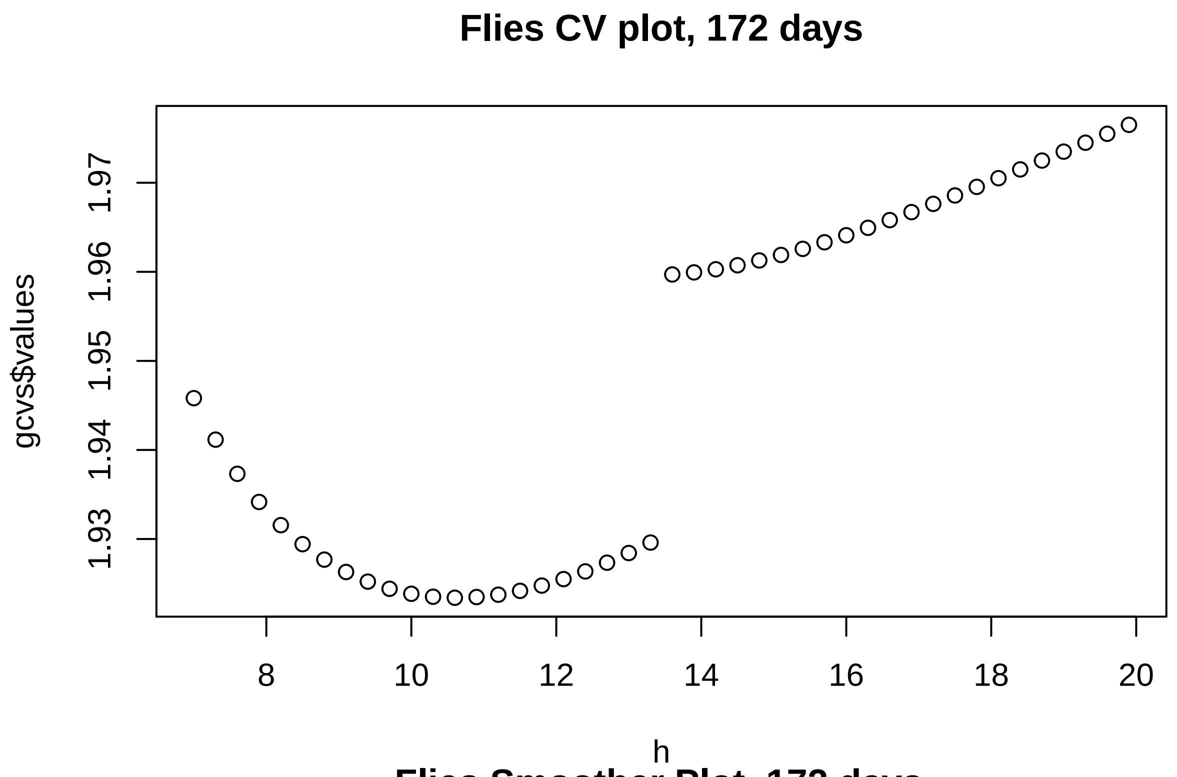


```
plot_smooth_bands(log_mort, flies$day, d[4], h_days_4, h_days_4)
```

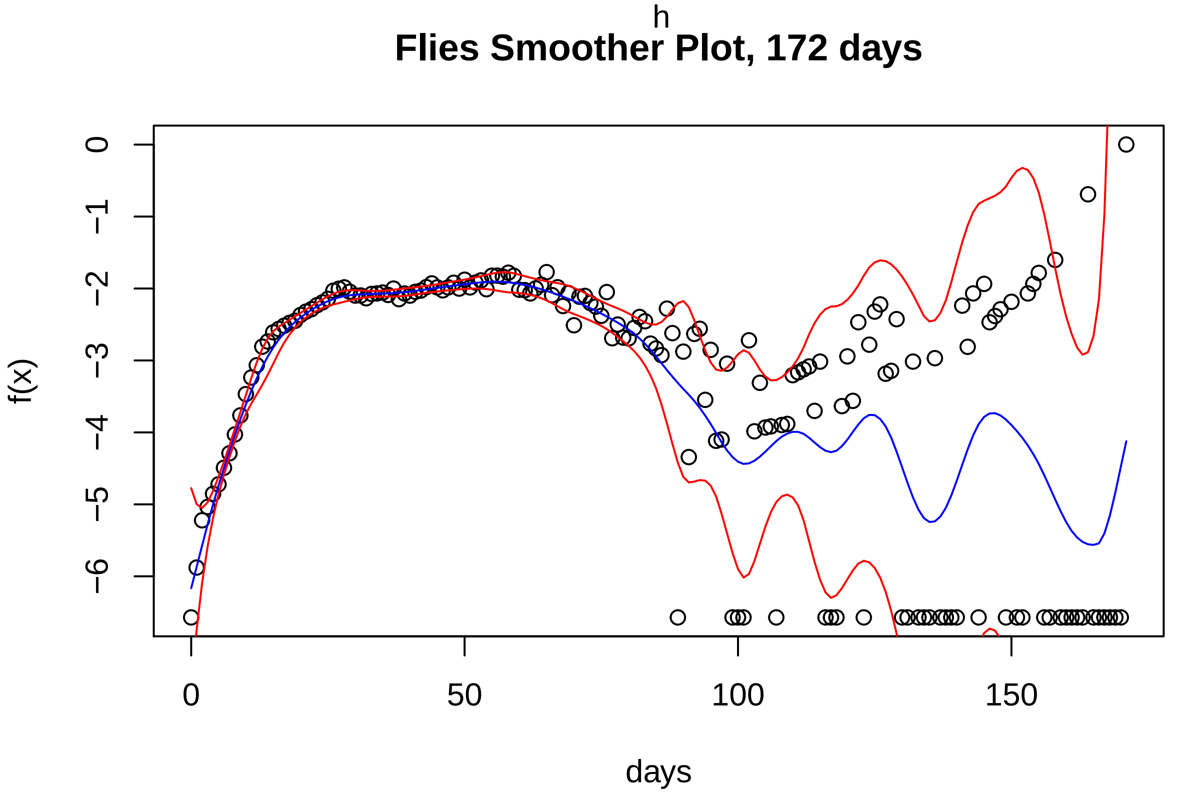




```r
plot_smooth_bands(log_mort, flies$day, d[5], h_days_5, h_days_5)
```



Flies CV plot, 172 days
gcvs$values
1.93
1.94
1.95
1.96
1.97
8
10
12
14
16
18
20
h



Flies Smoother Plot, 172 days
f(x)
0
−1
−2
−3
−4
−5
−6
0
50
100
150
days

c) Haar wavelets

estimating α

```r
J = 7
n = 2^J
t <- seq(1/n, 1, length.out = n)
log_mort_wave <- log_mort[1:n]

alpha_hat = mean(log_mort)
```

Estimating sigma, soft thresholding

```r
#Lafferty's Code for wavelet function (with level (j,k))
phi.h = function(x) { if ((x < 0) | (x > 1)) return(0); return(1)}
psi.h = function(x) { if ((x <= 0) | (x > 1)) return(0); if (x <= .5) return(-1); return(1) }

phi = function(x) {apply(as.matrix(x), 1, phi.h)}
psi = function(x) {apply(as.matrix(x), 1, psi.h)}

psi.jk = function(j,k,x) {2^{j/2} * psi((2^j)*x - k)}

#function to get Z_jk, assumes x[i] = i/n
#can also be used to return j level coefficents (vector) (just let k be 2^(j-1))
get_wavelet_coeff <- function(y, x, j, k){
  N = length(y)
  coeff = 0
  for(i in 1:N){
    coeff = coeff + (y[i] * psi.jk(j, k, x[i]))/ N
  }
  return(coeff)
}

beta.jk = function(j,k,f,N=1024) {
  coef = 0
  for (i in 1:N) {
    coef = coef + psi.jk(j,k,i/N) * f[i] * 1/N
  }
  return(coef)
}

res_level = J - 1
level_J_coeffs = get_wavelet_coeff(log_mort_wave, t, res_level, 0:(2^res_level - 1))
#lJB = beta.jk(res_level, 0:(2^res_level - 1), log_mort_wave, N = n)
#lJB == level_J_coeffs
print(sigma_hat <- mad(level_J_coeffs) ) #estimating sigma
```

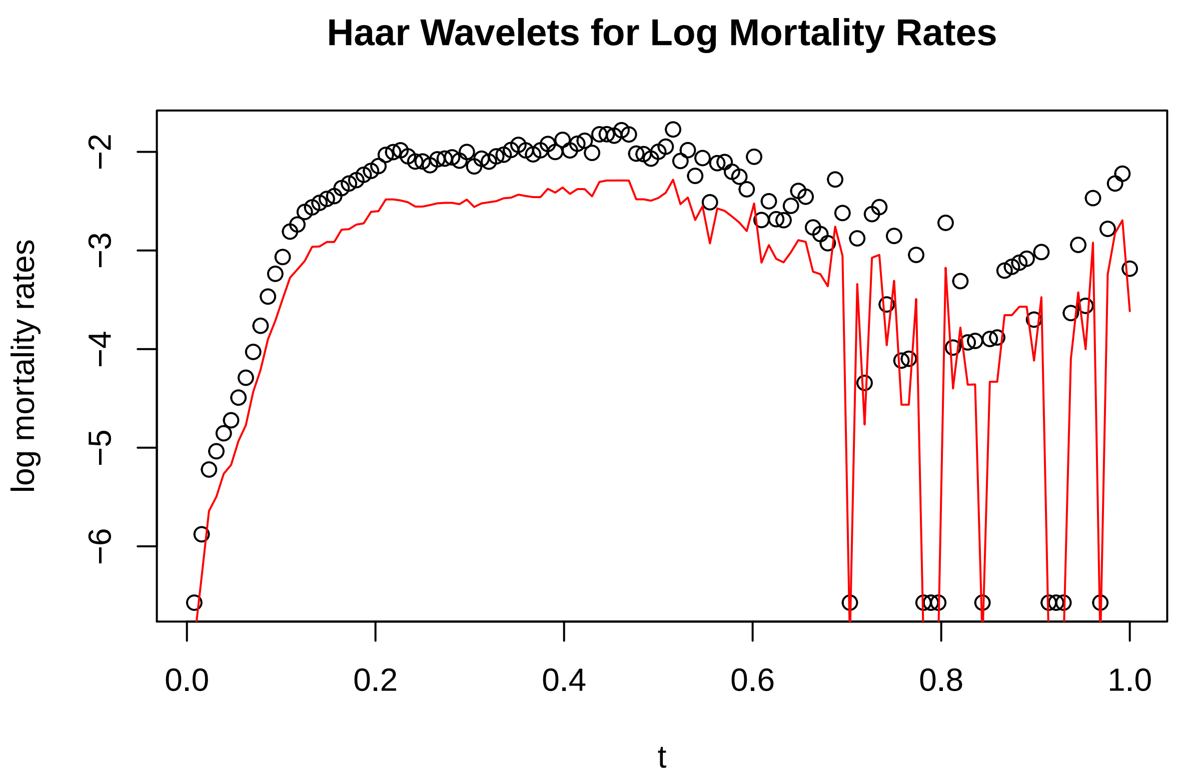
```
## [1] 0.009552436
```

```r
print(lambda <- sigma_hat * sqrt(2 * log(n) / n) ) #lambda param thresholding
```

```
## [1] 0.002630183
```

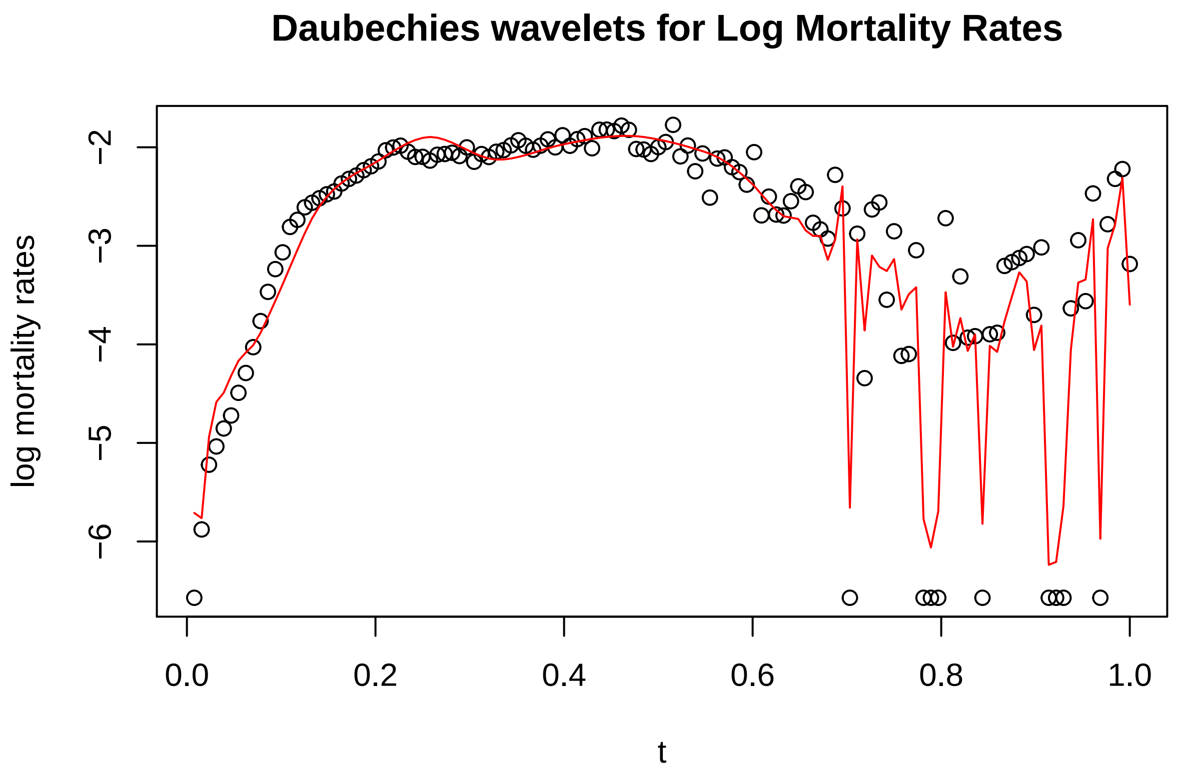
plotting)

```r
soft_threshold <- function(b, lambda){
  if(b > lambda){return(b - lambda)}
  if(b < -lambda){return(b + lambda)}
  return(0)
}

get_soft_threshold_coeffs <- function(D, lambda){
  thresholded_coeffs <- c()
  for(coeff in D){
    thresholded_coeffs <- c(thresholded_coeffs, soft_threshold(coeff, lambda) )
  }
  return(thresholded_coeffs)
}

compute_wavelet <- function(y, x, alpha_hat, res_level, lambda){
  fhat = apply(as.matrix(x), 1, function(x) {alpha_hat*phi(x)})
  for(j in 0:res_level) {
    beta = get_wavelet_coeff(y, x, j, 0:(2^j - 1) )
    beta_thres = get_soft_threshold_coeffs(beta, lambda)
    for(i in 1:length(x)) {
      fhat[i] = fhat[i] + sum(beta_thres * psi.jk(j,0:(2^j-1),x[i]))
      #fhat[i] = sum(beta_thres * psi.jk(j,0:(2^j-1),x[i]))
    }
  }
  return(fhat)
}

par(mfrow = c(1,1))
f_hat = compute_wavelet(log_mort_wave, t, alpha_hat, res_level, lambda)
plot(log_mort_wave ~ t, ylab = "log mortality rates", main = "Haar Wavelets for Log Mortality Rates")
lines(f_hat ~ t, col = "red")
```



d) Wavethresh Package

```
wmap_flies = wd(log_mort_wave, family="DaubLeAsymm", filter.number=8)
threshwmap_flies = threshold(wmap_flies, type="soft", policy="universal")
yhat_wf = wr(threshwmap_flies)
plot(log_mort_wave ~ t, ylab = "log mortality rates", main = " Daubechies wavelets for Log Mortality Ra
lines(yhat_wf ~ t, col = "red")
```


Daubechies wavelets for Log Mortality Rates
log mortality rates
−2
−3
−4
−5
−6
0.0
0.2
0.4
0.6
0.8
1.0
t